ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

FutJoin: Aplicación Web para organizar partidos de fútbol amateur

FutJoin: Web Application for amateur football matches schedulling

Realizado por
Javier Boned López

Tutorizado por
José Luis Pastrana Brincones

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, 24 DE JUNIO DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

En este trabajo de fin de grado se presenta una aplicación web destinada a permitir que los usuarios puedan organizar sus partidos de fútbol amateur, realizando la reserva de campos de fútbol de distintos complejos deportivos de Málaga. En el desarrollo de la aplicación destacará la omnipresencia de **Javascript**, plasmada en el *stack MEAN*, formado por las tecnologías ***MongoDB, Angular, Node.js y Express.*** De esta forma, los usuarios de la aplicación podrán consultar la disponibilidad de los campos en todo momento y, a través de una interfaz sencilla y cuidada, interactuar con ellos pudiendo crear partidos e invitar a otros jugadores. Cada usuario contará con un perfil donde ofrecerá tanto sus datos personales (nombre, email o fecha de nacimiento), como los futbolísticos (pierna buena, posición o altura) que podrá ser consultado por otro usuario de la aplicación.

Además, los complejos deportivos que colaboren con la aplicación ofrecerán una lista de sus campos disponibles, junto con sus características y el horario del que disponen.

Palabras clave: Angular, Node JS, Express, MongoDB, MEAN, API REST, aplicación web, base de datos, fútbol, complejos deportivos, partido.

Abstract

This end-of-degree project introduces a web application which allows to schedule your own amateur football games, booking from a variety of football pitches across sporting centers in Malaga. The development of the app is characterised by the homogeneity of ***JavaScript***, given expression in the *MEAN stack*, which is formed by technologies ***Mongo DB***, ***Angular***, ***Node.js*** and ***Express***. This way, application users would be able to check for the schedule availability of the fields at all times and, through a neat and simple graphic interface, interact with each other to organise games and invite new players. Each user will enjoy of a profile in which his personal details are listed (name, email, date of birth), as well as his footballing details (stronger foot, position of play or height) that can be seen by other users.

Furthermore, the sport centers that cooperate with the app will offer a list of all the available pitches, along with their characteristics and timetables.

Keywords: Angular, Node JS, Express, MongoDB, MEAN, API REST, Web Application, Database, football, soccer, sport center, football match.

Índice

RESUMEN **1**

ABSTRACT **1**

ÍNDICE **1**

INTRODUCCIÓN **1**

1.1 Motivación 1
1.2 Objetivos 2
1.3 Estructura de la memoria 3
1.4 Estado del arte 4
Timpik *4*
Biwenger *5*
1.5 Tecnologías utilizadas 5
1.5.1 JavaScript *5*
1.5.2 TypeScript *6*
1.5.3 MEAN Stack *6*
1.5.3.1 Angular 7
1.5.3.2 Node.js 7
1.5.3.3 Express 8
1.5.3.4 MongoDB 8
1.5.4 MagicDraw *8*
1.5.5 Visual Studio Code *9*
1.5.6 Angular Material *9*
1.5.7 JSON Web Token (JWT) *9*
1.5.8 BCrypt *9*

ANÁLISIS **10**

2.1 Obtención de requisitos 10
2.2 Análisis de requisitos 13
2.2.1 Iteración 1. Los usuarios. *14*
2.2.1.1 Requisitos funcionales 14
2.2.1.2 Requisitos no funcionales 15
2.2.1.3 Casos de Uso. 15
2.2.2 Iteración 2. Los complejos deportivos. *18*
2.2.2.1 Requisitos funcionales 18
2.2.2.2 Requisitos no funcionales 18
2.2.2.3 Casos de Uso. 19
2.2.3 Iteración 3. Los campos. *20*
2.2.3.1 Requisitos funcionales 21

2.2.3.2 Requisitos no funcionales 21
2.2.3.3 Casos de uso 22
2.2.4 Iteración 4. Los partidos. *24*
2.2.4.1 Requisitos funcionales 24
2.2.4.2 Requisitos no funcionales 25
2.2.4.3 Casos de uso 25
2.2.5 Requisitos no funcionales restantes *31*
2.3 ACTORES 31
2.4 DIAGRAMA DE CASOS DE USO. 32

DISEÑO 35

3.1 ARQUITECTURA SOFTWARE. 35
3.2 DISEÑO DEL SERVIDOR (*BACK-END*) 37
3.2.1 API REST *37*
3.2.2 Seguridad *38*
3.2.3 Base de datos *39*
3.3 DISEÑO DEL CLIENTE (*FRONT-END*) 42

IMPLEMENTACIÓN Y PRUEBAS 44

4.1. IMPLEMENTACIÓN DEL SERVIDOR. 45
4.1.1 JWT *45*
4.1.2 Funciones de la API REST *46*
4.1.3 Librerías utilizadas. *48*
4.2 IMPLEMENTACIÓN DEL CLIENTE 49
4.2.1 Componentes *50*
4.2.2 Servicios y Modelos *52*
4.2.3 Librerías utilizadas *52*
4.3. PRUEBAS. 53

CONCLUSIONES 55

5.1 CONCLUSIONES 55
5.2 LÍNEAS FUTURAS. 57

BIBLIOGRAFÍA 59

MANUAL DE USUARIO 61

1. PANTALLA DE INICIO 61
1.1 Inicio de sesión *61*
1.2 Registro de usuario *62*
1.3 Contacto de complejo *62*
2. HOME 63
2.1 Barra vertical izquierda *64*
2.2 Barra superior *65*
3. COMPLEJOS DEPORTIVOS 65
4. CAMPOS 66
5. PARTIDOS 67
5.1 Crear un partido *67*
5.2 Unirse a partido público *68*

5.3 Invitar a jugadores a partido. 69
5.4 Unirse a partido con invitación. 70
5.5 Convertir partido privado en público. 71
5.6 Cancelar partido 72
5.7 Cancelar asistencia a un partido 72

MANUAL DE INSTALACIÓN 73

1. REQUERIMIENTOS: 73
2. INSTALACIÓN DE SERVIDOR. 73
3. INSTALACIÓN DE CLIENTE. 74

1

Introducción

1.1 Motivación

El fútbol es un fenómeno que mueve multitudes. Según la Encuesta de Hábitos Deportivos de España publicada en 2015 por el CSD (Consejo Superior de Deportes), aproximadamente un cuarto de los españoles que realizan deporte, practica fútbol, en cualquier variedad [1]. Además, su repercusión internacional sigue aumentando. Según el Gran Censo (o Big Count), encuesta realizada entre las 207 asociaciones miembro de la FIFA (Federación internacional de Fútbol Asociación) en 2006, el número de personas que juegan al fútbol en todo el mundo es de 270 millones, siguiendo un ritmo ascendente.[2]

Por otro lado, el uso de aplicaciones web se ha convertido en algo habitual para la población. Y no solo con fines de ocio, sino para realizar actividades tan importantes como acceder a sus datos bancarios, o tan corrientes como hacer la compra. De esta forma, las aplicaciones web facilitan la vida de las personas, manteniéndolas comunicadas y creando interrelaciones entre ellas.

La cada vez más amplia demanda de estas origina la proliferación de nuevas técnicas y metodologías que optimicen y faciliten su uso para el usuario; de esta forma y con el objetivo de dar una experiencia más fluida a este, surgen las aplicaciones de **página única** o *simple-page-application*.

El concepto de aplicación de web de página única se fundamenta en una navegación sin necesidad de recargar el navegador, mostrando todas las pantallas en una misma página. Esto se consigue cargando todo el código al inicio de la aplicación de una sola vez, o consiguiente cargar dinámicamente recursos que la pagina vaya solicitando.

Relacionando los conceptos anteriores, es sorprendente que no exista una aplicación reputada que ayude a organizar partidos a la gran cantidad de gente que práctica este deporte de modo amateur. *FutJoin* surge de esta necesidad.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene por objeto el desarrollo de *FutJoin*, una aplicación web para permitir que los usuarios puedan organizar sus partidos de fútbol. La aplicación posibilitará a los mismos **crear y organizar sus partidos amateur** y englobará desde la reserva del campo donde se jugará hasta la confirmación de los jugadores interesados, con el fin de evitar al usuario realizar incómodas acciones cada vez que quiere organizar un partido de fútbol, como tener que llamar por teléfono al complejo deportivo en el que se encuentra el campo o tener que contactar de forma manual con cada uno de los jugadores que jugará el partido, por medio de plataformas como *WhatsApp* o *SMS*.

En cuanto a los complejos deportivos que gestionan los campos, el objetivo es que estos se promocionen y de una forma fluida, comuniquen a los usuarios a qué horas tienen libres sus campos y les ofrezcan la posibilidad de saber en todo momento la disponibilidad de ellos a través de una agenda con el fin de ganar fluidez en la comunicación entre cliente y complejo deportivo.

Por último, cabe recalcar también como objetivo lograr la seguridad en los diferentes campos que la aplicación abarca:

- Datos de la aplicación Web: Garantizar la seguridad de los datos personales de los usuarios, así como de sus contraseñas.
- Base de datos: Garantizar la seguridad de los datos que contenga la base de datos.
- Comunicación complejo-usuario: Garantizar la seguridad en la comunicación entre un complejo deportivo y un usuario, asegurándo de que los datos lleguen correctamente a las dos partes.

1.3 Estructura de la memoria

La memoria se ha estructurado de forma que el lector pueda apreciar la evolución del desarrollo de la aplicación, transitando por las distintas etapas de desarrollo software que esto conlleva. Dividida en capítulos, el contenido de la memoria es el siguiente:

Capitulo 1: Introducción. En este capítulo se enuncia un breve prólogo sobre lo que va a ser el proyecto. Comienza por la motivación que ha llevado al desarrollo de este, continúa por los objetivos detallados, analiza las diferentes alternativas que se pueden encontrar para abordar el problema a solucionar, muestra una breve estructura de lo que es toda la memoria y expone de manera detallada las tecnologías utilizadas a lo largo de el proyecto.

Capítulo 2. Análisis. En el segundo capítulo se describe con más detalle los requisitos que se obtienen al estudiar las necesidades de los usuarios. De esta manera, se identifican los requisitos funcionales y no funcionales, se realiza una breve descripción sobre ellos y se reconocen los actores que van a interactuar con el sistema. A raíz de este análisis, se originan y se detallan los casos de uso a contemplar.

Capítulo 3. Diseño. En el tercer capítulo se profundiza en la arquitectura del sistema, tanto en la parte del servidor describiendo la API REST que se va a desarrollar para generar servicios, como en la del cliente, así como en las técnicas utilizadas. Además, se mostrará el diagrama de clases elaborado para diseñar la base de datos, así como las primeras decisiones tomadas relacionadas tanto en la interfaz (los colores principales y el logo) como en la seguridad (la autenticación de los usuarios).

Capítulo 4: Implementación y pruebas. El cuarto capítulo explica pormenorizadamente el funcionamiento e implementación de los diferentes componentes de la aplicación. Se presentan diferentes capturas de la interfaz de la aplicación, explicando el porqué de las decisiones tomadas. Además, recoge varios errores surgidos a lo largo del desarrollo, junto a las soluciones implementadas.

Capítulo 5: Conclusiones. El último capítulo está dedicado al resultado final del proyecto, confrontando el mismo con los objetivos principales. Se exponen las conclusiones personales obtenidas a lo largo del proceso de desarrollo, así como los conocimientos adquiridos en el mismo. Contempla también las posibles líneas futuras del proyecto.

Por último, al final de la memoria se reseñará la bibliografía y dos anexos: un manual de usuario para dar asistencia a las personas que quieran utilizar la aplicación y un manual de instalación para poder ejecutar el proyecto en cualquier sistema.

1.4 Estado del arte

Se van a analizar diferentes proyectos que cubren necesidades similares a la de la aplicación desarrollada. Algunos de ellos coinciden casi totalmente con esta y otros comparten ciertas características, como el fútbol o la reserva de un determinado horario.

Timpik

Timpik es una aplicación que permite al usuario organizar actividades deportivas y encontrar compañeros para realizarlas, de forma que no dejemos de practicar deporte por no tener quien, cuando y donde.[3]
El principal problema de esta aplicación es una interfaz web no del todo amigable, junto al escaso numero de usuarios que tiene en la ciudad de Málaga, en la que no se puede encontrar fácilmente un partido de fútbol con jugadores suficientes. Además, la aplicación está destinada a todo tipo de deportes, no únicamente al fútbol, lo que provoca que no tenga un gran público especializado en un deporte, sino usuarios repartidos en todos ellos.

Biwenger

Biwenger es un simulador de fútbol en el que el usuario dirige un equipo compuesto por jugadores reales en una liga virtual y compite con otros usuarios para ganarla. Se nutre de un sistema de puntuación en el que los jugadores consiguen mejor valoración conforme a sus actuaciones en la vida real en la competición que participan.[4]

Aunque *Biwenger* no está desarrollado con el mismo fin que *Futjoin*, se pueden encontrar muchas similitudes entre ellos. En primer lugar, el fútbol es el tema principal sobre el que se basa todo el desarrollo, cuidando una interfaz compuesta por colores relacionados con este deporte, así como la utilización de iconos que llamen la atención del usuario.
Además, en este caso a diferencia del anterior, *Biwenger* sí cuenta con un gran numero de jugadores, lo que invita a pensar que centrar el desarrollo en un publico determinado (en este caso, el del fútbol) puede ser indicio de un gran éxito.

1.5 Tecnologías utilizadas

En la siguiente tabla se van a describir las principales herramientas, lenguajes de programación, *frameworks* y librerías utilizadas para este proyecto. Las que se consideren más importantes se explicarán mas detenidamente.

1.5.1 JavaScript

JavaScript es un lenguaje de programación interpretado con capacidades orientadas a objetos (OO). Normalmente se utiliza en los navegadores web y logra extender su funcionalidad permitiendo la creación con objetos que permiten a los *scripts* interactuar con el usuario, controlar el navegador web y modificar el contenido del de la propia web.[5]

1.5.2 TypeScript

TypeScript es una extensión de JavaScript destinada a permitir un desarrollo más fácil de aplicaciones a gran escala implementadas en JavaScript. Esto significa que un programa JavaScript es a la vez un programa TypeScript, pero no al revés, ya que este último ofrece un sistema de módulos, clases e interfaces más rico. La intención de esta extensión es que la transición no suponga ningún problema a un programador de JavaScript.[6]

1.5.3 MEAN Stack

El *stack MEAN* es una potente herramienta web formada por 4 grandes bloques que comparten una misma característica: el lenguaje en el que están desarrollados es JavaScript. Estas 4 tecnologías son: *MongoDB* como base de datos, *Express* como *framework* del servidor web, Node.js como plataforma principal del servidor y Angular como *framework* del cliente web.[7]

Las grandes ventajas que proporciona esta herramienta son las siguientes:

- Consigue crear una aplicación isomórfica, es decir, una aplicación en la que todas sus capas están programadas con el mismo lenguaje. Esto la hace fácilmente manejable, modificable y entendible de cara al programador.
- El manejo de datos se realiza con JSON, formato de texto sencillo para el intercambio de datos.
- Se adapta especialmente bien a la arquitectura basada en API REST, que, unida al formato de datos JSON anteriormente descrito, produce un flujo de información ligero, rápido y potente.

A continuación vamos a analizar cada uno de las herramientas que la componen.

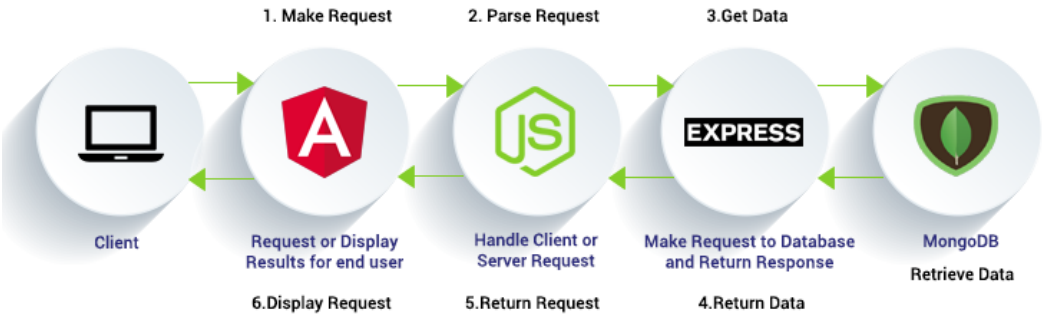


Figura 1 Como trabaja el *stack MEAN* y que labor desempeña cada tecnología que la compone. Fuente: https://www.chromeinfotech.net/blog/wp-content/uploads/2019/01/v3-MEAN-Stack-Modern-approach.png

1.5.3.1 Angular

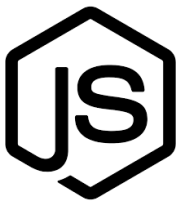
Angular es un *framework* de *front-end* impulsado por *Google*. Se utiliza para crear aplicaciones web de una sola página y está desarrollado en *TypeScript*.[8]

Angular está formado por bloques llamados componentes, que actúan de forma independiente, conteniendo cada uno sus propias funcionalidades, página HTML y estilo específico.

Aunque Angular está integrado con la arquitectura original del estilo *Modelo-Vista-Controlador* (MVC), la realidad es que no cumple todas sus normas.

Una aplicación *Angular* está compuesta por un árbol de componentes, en el cual cada uno cumple una función específica.

1.5.3.2 Node.js



Node.js está concebido como un entorno de ejecución de código abierto basado en ECMAScript (lenguaje construido sobre *JavaScript*) orientado a eventos asíncronos diseñado para ser de gran utilidad de cara a la construcción de aplicaciones en red altamente escalables.[9] Esto quiere decir que está preparado para soportar una gran cantidad de conexiones simultáneas, lo que lo convierte en una plataforma ideal para desarrollar el servidor de una aplicación web.

1.5.3.3 Express

Express es el *framework* más popular paraaplicaciones web construidas en Node.js y proporciona un enorme conjunto de mecanismos y funciones. Entre todas las ventajas que aporta, cabe destacar los métodos relacionados con el protocolo HTTP que permiten la realización de una API sólida de manera sencilla y la posibilidad de procesar peticiones "middleware" adicionales que aportan una gran flexibilidad a la aplicación, pudiendo confrontar cualquier tipo de problema de desarrollo web.[10]

1.5.3.4 MongoDB

MongoDB es un moderno sistema de base de datos *NoSQL* de código abierto. Se trata de una base de datos documental, es decir, almacena y gestiona datos estructurados en documentos que no se ajustan a un esquema estándar como en una base de datos relacional. *MongoDB* guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, de manera que al no existir una estructura fija, se consigue una integración de los datos más fácil y rápida (en ciertas aplicaciones).[11]

Una vez terminado el análisis de las principales tecnologías utilizadas, cabe nombrar y analizar brevemente algunos programas/librerías o herramientas utilizadas a lo largo de la elaboración del proyecto, así como a funciones o estándares empleados para lograr la seguridad en él.

1.5.4 MagicDraw

MagicDraw es una herramienta CASE utilizada para realizar los diagramas de la aplicación usando la notación UML.

1.5.5 Visual Studio Code

Editor de código desarrollado por *Microsoft* optimizado para desarrollar aplicaciones web modernas. Es gratuito y de código abierto.

1.5.6 Angular Material

Angular Material es una librería para *Angular* compuesta por componentes cuyo único fin es mejorar el diseño de la aplicación. Cuenta con un gran repertorio de componentes para hacer la interfaz más amigable con el usuario.

1.5.7 JSON Web Token (JWT)

Estándar basado en JSON para proporcionar seguridad a los datos enviados entre aplicaciones o servicios, garantizando su validez y su consistencia. Esto se logra a través de *tokens* de acceso. En el caso de la aplicación desarrollada en el trabajo, se utilizará para autenticar e identificar a los usuarios así como conceder los privilegios oportunos.

1.5.8 BCrypt

Función de *hashing* de contraseñas basada en el cigrado de BlowFish. Lleva incorporado un fragmento aleatorio llamado *salt* que usa para generar el hash asociado a la contraseña. Así, evita que con dos contraseñas iguales se genere el mismo hash. [12]

2

Análisis

2.1 Obtención de requisitos

Para obtener información sobre el tipo de usuario que iba a hacer uso de la aplicación y sobre las necesidades que precisaba, se realizó una encuesta a través de *Google Forms*. En ella participaron 53 usuarios, que aunque no sea una muestra significativa, contribuyeron a la toma de algunas decisiones.
En ella se hicieron las siguientes preguntas:

1. ¿Qué edad tienes?
    - Menos de 18.
    - 18 – 30 años.
    - 30 – 50 años.
    - Más de 50 años.

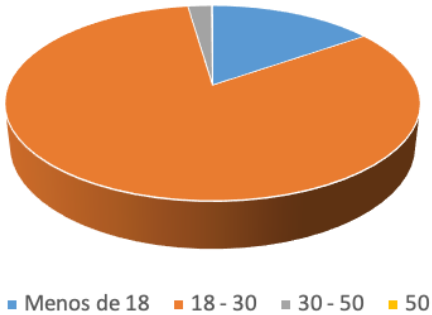


Figura 2. Estadística encuesta pregunta 1.

Esta cuestión se formuló para saber a qué público se iba a dirigir la aplicación, si a uno más joven en el que se pudiera utilizar colores más llamativos y una interfaz menos formal o a un público de más edad, en el que habría que extremar el cuidado de la interfaz.
Como se puede ver en la Figura 2, el resultado fue el esperado: el público de la aplicación en general iba a ser muy joven y por lo tanto, la interfaz debía ser apropiada para este.

2. ¿Con qué frecuencia juegas al fútbol?
    - 1-2 veces por semana.
    - 3-4 veces por semana.
    - Más de 4 veces por semana.

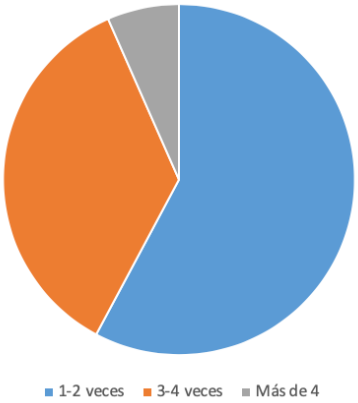


Figura 3. Estadística encuesta pregunta 2.

Esta cuestión se formuló para conocer la frecuencia con la que los usuarios practican este deporte. Su utilidad consiste en constatar el número de prácticas deportivas que tienen los usuarios para comprobar si realmente la aplicación viene a cubrir una necesidad existente. La respuesta, aunque con una incidencia menor de la prevista, deja patente que los usuarios practican el deporte, con más o menos regularidad.

3. ¿Y con qué frecuencia te gustaría jugar?

    - 1-2 veces por semana.
    - 3-4 veces por semana.
    - Más de 4 veces por semana.

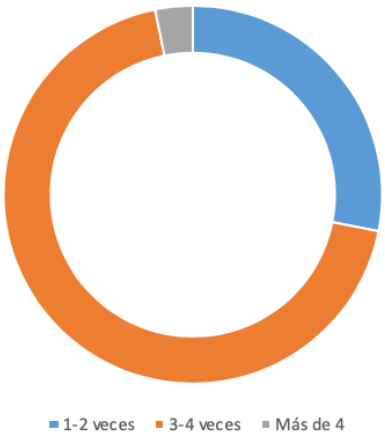


Figura 4. Estadística encuesta pregunta 3.

Esta cuestión es clave para comprobar realmente si la necesidad sobre la que se fundamenta la idea de *Futjoin* era real. Y, afortunadamente, la confirmó. En general, a los usuarios les gustaría jugar más de lo que juegan.

4. ¿Cómo sueles organizar con tus amigos los partidos de fútbol?

    - Whatsapp
    - Correo electrónico
    - Llamadas de teléfono
    - Otros

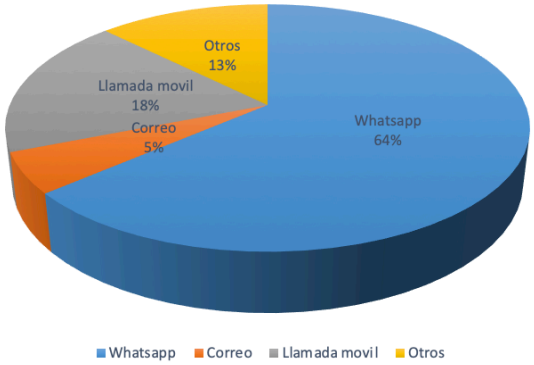


Figura 5. Estadística encuesta pregunta 4.

A partir de la pregunta 4 hay un cambio de objetivo. Ya localizado el tipo de usuario al que se va a dirigir la aplicación, había que centrarse en la reserva del campo y en la organización de los partidos. Con esta pregunta se verifica que no hay una plataforma en la que se puedan realizar estas acciones, si no que los usuarios acuden habitualmente a sistemas de mensajería instantánea como *Whatsapp* o al correo electrónico.

5. ¿Te gustaría poder reservar las pistas de fútbol a través del ordenador?

    - Sí
    - No

En la consulta número 5 se confirma que el uso de aplicaciones web está absolutamente normalizado, ya que los usuarios responden "sí" en un 87,5% de los casos, manifestando que no les supone ningún esfuerzo adicional utilizar el ordenador para reservar una pista de fútbol.

6. ¿Tienes problemas para contactar y confirmar la asistencia de los jugadores al partido?

    - Sí
    - No

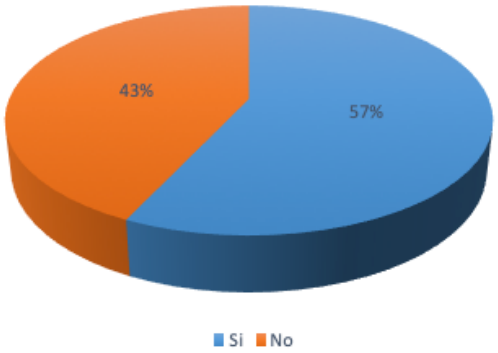


Figura 6. Estadística encuesta pregunta 6

Esta última consulta se suscita debido a una cuestión surgida al crear y modelar un partido. Se partía de una premia: permitir la creación de partidos públicos para que cualquier usuario sin invitación se pudiera unir a los mismos, solucionando así el posible inconveniente del insuficiente número de jugadores en un partido. El resultado ratificaba la existencia del problema, que posteriormente tendría una gran repercusión a la hora del desarrollo de la aplicación.

Gracias a esta encuesta se pudieron solucionar algunas cuestiones surgidas y tomar ciertas decisiones para el posterior análisis de requisitos.

2.2 Análisis de requisitos

Antes de empezar a listar los requisitos funcionales y no funcionales, cabe destacar la metodología que se ha utilizado para el desarrollo del proyecto, ya que tiene influencia directa sobre ellos. La metodología empleada ha sido una **metodología iterativa de desarrollo** cuyo análisis y diseño han estado basados en el **Proceso Unificado**. Esta se fundamenta en agrupar las tareas en distintos bloques temporales denominados **iteraciones**. En esta metodología, al final de cada iteración, cada requerimiento debe tener una completa implementación, de modo que en cada una de ellas los componentes logren desarrollar progresivamente el producto dependiendo de los completados en las iteraciones antecesoras.[13]

De esta forma, la obtención y análisis de requisitos se ha realizado dividida en iteraciones y por lo tanto, se van a describir en el mismo orden que se ha utilizado en el desarrollo. A su vez, también se detallarán los casos de uso dentro de cada bloque.

Los 4 bloques en los que se han dividido los requisitos son: el bloque de los **usuarios**, el de **los complejos deportivos**, el de **los campos de fútbol** y el de **los partidos**.

En cada bloque se analizarán los requisitos funcionales, que son los que determinan la funcionalidad de la aplicación y definen las acciones que el sistema debe cumplir y los no funcionales, que especifican restricciones o condiciones sobre las que se ejecuta el sistema.

2.2.1 Iteración 1. Los usuarios.

La primera iteración se encarga del que posiblemente sea factor más importante de la aplicación: los usuarios. Un usuario será cualquier persona que tenga acceso a la aplicación y será el que a través de sus decisiones, genere las acciones del sistema
.

2.2.1.1 Requisitos funcionales

Identificador	Nombre	Descripción
RF1	Inicio de sesión	Los usuarios registrados deben poder autenticarse en la aplicación.
RF2	Cierre de sesión	Los usuarios deben poder salir de aplicación cerrando sus sesión.
RF3	Realizar registro	Los usuarios que no estén registrados en la aplicación deben poder registrarse.
RF4	Actualizar usuario	Los usuarios registrados en la aplicación deben poder actualizar sus datos siempre que lo deseen.

2.2.1.2 Requisitos no funcionales

Identificador	Nombre	Descripción
RNF1	Seguridad en la contraseña	La aplicación debe ser capaz de almacenar las contraseñas de forma segura, por medio de algún método que las asegure.
RNF2	Seguridad en los datos del usuario	El usuario deberá estar identificado de forma segura durante toda la sesión.
RNF3	Documentación	La aplicación deberá contener un manual de usuario donde indique los pasos para la correcta gestión del usuario.

2.2.1.3 Casos de Uso.

Nombre	Descripción
Caso de uso	Inicio de sesión
Actores	Usuario estándar
Descripción	Los usuarios registrados deben poder autenticarse en la aplicación.
Precondiciones	El usuario no está registrado en la aplicación
Escenario Principal	1. El usuario introduce su email y su contraseña. 2. El usuario accede a la página inicial de la aplicación
Escenario alternativo	1.b. El usuario introduce sus datos de forma incorrecta. 2.b El usuario recibe un aviso de que los datos son incorrectos y permanece en la página de *login*.

Nombre	Descripción
Caso de uso	Cierre de sesión
Actores	Usuario estándar
Descripción	El usuario deberá poder cerrar su sesión una vez iniciada está.
Precondiciones	El usuario está ha iniciado sesión en la aplicación
Escenario Principal	1. El usuario accede al elemento de cierre de sesión situado en el marco de la aplicación. 2. El usuario es redirigido a la página de *login*.
Escenario alternativo	No existe escenario alternativo.

Nombre	Descripción
Caso de uso	Realizar registro
Actores	Usuario estándar
Descripción	El usuario que no esté registrado en a aplicación deberá poder hacerlo a través de un formulario adecuado.
Precondiciones	El usuario no está registrado en la aplicación
Escenario Principal	1. El usuario accede a "Registro de usuario" en la pantalla de inicio de la aplicación. 2. El usuario introduce el email con el que se va a registrar. 3. El usuario introduce la contraseña con la que se va a registrar. 4. El usuario introduce distintos datos futbolísticos. 5. El usuario introduce distintos datos personales. 6. El usuario confirma el registro y recibe un aviso de que el registro se ha realizado correctamente.

Escenario alternativo	2.b El email que se introduce ya existe. 2.c El sistema muestra al usuario un aviso de que debe cambiar el email. 3.b. La contraseña no cumple con la seguridad requerida por la aplicación. 3.c El sistema muestra al usuario un aviso de que la contraseña debe ser más segura. 4.b Los datos no cumplen con las restricciones del sistema. 4.c El sistema muestra al usuario un aviso de que los datos deben cumplir las restricciones del sistema.

Nombre	Descripción
Caso de uso	Actualizar usuario
Actores	Usuario estándar
Descripción	El usuario deberá poder actualizar sus datos una vez registrado.
Precondiciones	El usuario está ha iniciado sesión en la aplicación
Escenario Principal	1. El usuario accede a “mis datos” 2. El sistema muestra los datos del usuario hasta el momento, permitiendo al usuario modificarlos. 3. El usuario modifica sus datos a su voluntad. 4. El usuario confirma los nuevos datos.
Escenario alternativo	3.b El usuario modifica los datos sin satisfacer las restricciones del sistema. 3.c El sistema informa al usuario de que debe cumplir las condiciones impuestas.

2.2.2 Iteración 2. Los complejos deportivos.

Los complejos deportivos son la otra parte fundamental de la aplicación junto a los usuarios, ya que gracias a ellos se podrá acceder a los distintos campos de fútbol existentes y a la vez, a los partidos que se organicen en estos.

2.2.2.1 Requisitos funcionales

Identificador	Nombre	Descripción
RF5	Crear complejo deportivo	El usuario administrador podrá crear nuevos complejos deportivos que colaboren con la aplicación.
RF6	Eliminar complejo deportivo	El usuario administrador podrá eliminar los complejos deportivos.

Como ya se ha expuesto anteriormente, en la metodología utilizada los componentes de las iteraciones logran evolucionar el producto sobre las iteraciones anteriores. Por esto, cabe destacar que aunque los requisitos estén ligados al bloque de los complejos deportivos, van a ser los usuarios los que realicen ciertas acciones, como las siguientes:

Identificador	Nombre	Descripción
RF7	Visualizar complejos deportivos	Los usuarios registrados deben poder visualizar los complejos deportivos que estén registrados en la aplicación.

2.2.2.2 Requisitos no funcionales

Los identificadores de los requisitos no van a cambiar respecto a la anterior iteración, porque, aunque los datos del complejo y del usuario sean distintos, el requisito no funcional que los engloba es el mismo. En el resto de iteraciones en que se planteé esta situación, se aplicará el mismo método.

Identificador	Nombre	Descripción
RNF2	Seguridad en los datos del complejo	La aplicación debe ser capaz de almacenar los datos de los complejos deportivos que colaboren con la aplicación de forma segura.
RNF3	Documentación	La aplicación deberá contener un manual de usuario donde indique los pasos para el correcto tratamiento de los complejos deportivos.

2.2.2.3 Casos de Uso.

Nombre	Descripción
Caso de uso	Crear complejo deportivo
Actores	Usuario administrador.
Descripción	El usuario administrador podrá crear nuevos complejos deportivos que colaboren con la aplicación.
Precondiciones	El complejo deportivo no está registrado en la aplicación.
Escenario Principal	1. El complejo deportivo se pone en contacto con la aplicación mediante un formulario en la pagina de inicio de la aplicación. 2. El usuario administrador inicia sesión en la aplicación. 3. El usuario administrador crea el complejo con los datos correspondientes.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Eliminar complejo deportivo
Actores	Usuario administrador.
Descripción	El usuario administrador podrá eliminar los complejos deportivos de aplicación.
Precondiciones	1. El complejo deportivo está registrado en la aplicación.
Escenario Principal	1. El usuario administrador elimina el complejo deportivo de la aplicación.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Visualizar complejos deportivos
Actores	Usuario estándar
Descripción	Los usuarios dela aplicación podrán visualizar los diferentes complejos deportivos.
Precondiciones	1. El complejo deportivo está registrado en la aplicación. 2. El usuario está registrado y ha iniciado sesión en la aplicación.
Escenario Principal	1. El usuario accede al menú correspondiente de los complejos deportivos.
Escenario alternativo	No existe escenario alternativo.

2.2.3 Iteración 3. Los campos.

Dentro de los complejos deportivos encontramos los campos, los cuales incluyen un calendario con los días y horas disponibles para organizar partidos.

2.2.3.1 Requisitos funcionales

Identificador	Nombre	Descripción
RF8	Crear Campo	El complejo deportivo podrá crear nuevos campos.
RF9	Eliminar Campo	El complejo deportivo podrá eliminar sus campos.
RF10	Actualizar campo	El complejo deportivo podrá actualizar los datos de sus campos.
RF11	Visualizar campos	Los usuarios verán los campos de los complejos deportivos registrados en la aplicación
RF12	Visualizar horario.	Los usuarios visualizarán los horarios disponibles de los campos, así como los horarios ocupados.

2.2.3.2 Requisitos no funcionales

Identificador	Nombre	Descripción
RNF2	Seguridad en los datos de los campos	La aplicación debe ser capaz de almacenar los datos de los campos correctamente.
RNF3	Documentación	La aplicación deberá contener un manual de usuario donde indique los pasos para el correcto tratamiento de los campos.
RNF8	Mantenimiento de calendario	La aplicación deberá mostrar un calendario con los horarios disponibles de forma intuitiva y fácil para el usuario.

2.2.3.3 Casos de uso

Nombre	Descripción
Caso de uso	Crear campo
Actores	Usuarios complejos deportivos.
Descripción	Los complejos deportivos podrán crear nuevos campos.
Precondiciones	El complejo deportivo está registrado en la aplicación.
Escenario Principal	1. El complejo deportivo accede a sus datos. 2. El complejo deportivo crea un nuevo campo. 3. El complejo deportivo introduce los datos necesarios para crear un nuevo campo. 4. El complejo deportivo crea el campo.
Escenario alternativo	3.b. Los datos introducidos del campo no cumplen los requisitos necesarios de la aplicación 3.c. El sistema muestra una alarma informando de que se deben cumplir las restricciones.

Nombre	Descripción
Caso de uso	Eliminar campo
Actores	Usuarios complejos deportivos.
Descripción	Los complejos deportivos podrán eliminar sus campos.
Precondiciones	1. El complejo deportivo está registrado en la aplicación.
Escenario Principal	1. El complejo deportivo accede a sus datos. 2. El complejo deportivo elimina el campo que crea conveniente.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Actualizar campos
Actores	Usuarios complejos deportivos.
Descripción	Los complejos deportivos podrán actualizar los datos de sus campos
Precondiciones	1. El complejo deportivo está registrado en la aplicación. 2. El complejo deportivo tiene al menos un campo para modificar.
Escenario Principal	1. El complejo deportivo accede a sus datos. 2. El complejo deportivo modifica los datos de un campo creado. 3. El complejo deportivo actualiza el campo.
Escenario alternativo	2.b El usuario modifica los datos sin satisfacer las restricciones del sistema. 2.c El sistema informa al usuario de que debe cumplir las condiciones impuestas.

Nombre	Descripción
Caso de uso	Visualizar campos
Actores	Usuarios.
Descripción	Los usuarios de la aplicación podrán visualizar los diferentes campos de los complejos deportivos.
Precondiciones	1. El complejo deportivo está registrado en la aplicación. 2. El usuario está registrado y ha iniciado sesión en la aplicación. 3. El complejo deportivo tiene al menos un campo.

Escenario Principal	1. El usuario accede al menú correspondiente de los complejos deportivos. 2. El usuario accede a los campos del complejo deportivo elegido.
Escenario alternativo	No existe escenario alternativo

2.2.4 Iteración 4. Los partidos.

Los partidos conforman el último gran bloque del análisis. Se relacionan con todos los anteriores, ya que con ellos interactúan los usuarios, los complejos deportivos y los campos en los que se van a disputar.

2.2.4.1 Requisitos funcionales

Identificador	Nombre	Descripción
RF13	Crear partido público	Los usuarios podrán crear sus propios partidos públicos en horarios libres de los campos.
RF14	Unirse a partido público	Los usuarios podrán unirse a los partidos públicos creados.
RF15	Crear partido privado	Los usuarios podrán crear sus propios partidos privados en horarios libres de campos.
RF16	Cambiar partido privado a público	Los usuarios creadores de partidos podrán convertir un partido privado en público.
RF17	Invitar jugadores.	Los usuarios creadores de partidos podrán invitar a otros jugadores a participar en su partido.
RF18	Cancelar partido	Los usuarios creadores de partidos podrán cancelar el partido.
RF19	Cancelar asistencia a partido	Los usuarios que participen en los partidos podrán cancelar su asistencia al partido.

RF20	Unirse a partido privado	Los usuarios que reciban invitaciones para un partido privado podrán aceptarlas y se unirán al partido creado.
RF21	Ocupar partido	Los complejos deportivos podrán ocupar las horas que deseen, eliminado su disponibilidad de cara a los usuarios.

2.2.4.2 Requisitos no funcionales

Identificador	Nombre	Descripción
RNF2	Seguridad en los datos	La aplicación debe ser capaz de almacenar los datos de los partidos correctamente.
RNF3	Documentación	La aplicación deberá contener un manual de usuario donde indique los pasos para la correcta organización de los partidos.
RNF4	Mantenimiento de calendario	La aplicación deberá mostrar un calendario con los partidos creados.
RNF5	Invitaciones de usuarios	La aplicación debe controlar las invitaciones de los partidos de los usuarios y facilitar una correcta comunicación entre usuarios.
RNF6	Diferenciar entre partido público y privado.	La aplicación debe ser capaz de manifestar de forma clara la diferencia entre los partidos públicos y privados.

2.2.4.3 Casos de uso

Nombre	Descripción
Caso de uso	Crear partido público
Actores	Usuario estándar
Descripción	Los usuarios de la aplicación podrán crear sus propios partidos públicos

Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El complejo deportivo está registrado en la aplicación. 3. El complejo tiene al menos un campo.
Escenario Principal	1. El usuario accede al menú correspondiente de los complejos deportivos. 2. El usuario accede a los campos del complejo deportivo elegido. 3. El usuario elige el campo en el que quiera hacer la reserva. 4. El usuario elige el horario en el que quiere crear el partido. 5. El usuario escoge “público” como tipo del partido.
Escenario alternativo	4.b El horario elegido ya está ocupado. 4.c El sistema no permitirá al usuario realizar esa acción.

Nombre	Descripción
Caso de uso	Unirse a partido público
Actores	Usuario estándar
Descripción	Los usuarios podrán unirse a los partidos públicos creados.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. Existe un partido público
Escenario Principal	1. El usuario accede al menú correspondiente de los complejos deportivos. 2. El usuario accede a los campos del complejo deportivo elegido.

	3. El usuario elige el campo en el que quiera hacer la reserva. 4. El usuario elige el horario en el existe un partido público. 5. El usuario decide unirse al partido..
Escenario alternativo	5.b El partido elegido ya está lleno. 5.c El sistema no permitirá al usuario unirse y le informará de ello.

Nombre	Descripción
Caso de uso	Crear partido privado
Actores	Usuario estándar
Descripción	Los usuarios podrán crear sus propios partidos privados en horarios libres de campos..
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El complejo deportivo está registrado en la aplicación. 3. El complejo tiene al menos un campo.
Escenario Principal	1. El usuario accede al menú correspondiente de los complejos deportivos. 2. El usuario accede a los campos del complejo deportivo elegido. 3. El usuario elige el campo en el que quiera hacer la reserva. 4. El usuario elige el horario en el que quiere crear el partido. 5. El usuario escoge “privado” como tipo del partido.
Escenario alternativo	4.b El horario elegido ya está ocupado. 4.c El sistema no permitirá al usuario realizar esa acción.

Nombre	Descripción
Caso de uso	Cambiar partido privado a público
Actores	Usuario estándar
Descripción	Los usuarios creadores de partidos podrán convertir un partido privado en público.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El usuario ha creado un partido privado.
Escenario Principal	1. El usuario accede al menú correspondiente de sus partidos. 2. El usuario elige el partido privado creado y lo cambia a público.
Escenario alternativo	No existe escenario alternativo.

Nombre	Descripción
Caso de uso	Invitar jugadores.
Actores	Usuario estándar
Descripción	Los usuarios creadores de partidos podrán invitar a otros jugadores a participar en su partido.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El usuario ha creado algún partido.
Escenario Principal	1. El usuario accede al menú correspondiente de sus partidos. 2. El usuario elige el partido creado. 3. El usuario selecciona los usuarios que quiere invitar. 4. El sistema envía invitaciones a los usuarios seleccionados.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Cancelar partido
Actores	Usuario estándar
Descripción	Los usuarios creadores de partidos podrán cancelar el partido creado.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El usuario ha creado algún partido.
Escenario Principal	1. El usuario accede al menú correspondiente de sus partidos. 2. El usuario elige el partido creado. 3. El usuario cancela el partido creado. 4. El sistema informa a todos los usuarios jugadores del partido de que el partido ha sido eliminado.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Cancelar asistencia a partido
Actores	Usuario estándar
Descripción	Los usuarios que participen en los partidos podrán cancelar su asistencia al partido.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación. 2. El usuario participa en algún partido.
Escenario Principal	1. El usuario accede al menú correspondiente de sus partidos. 2. El usuario elige el partido al que quiere cancelar la asistencia. 3. El usuario cancela la asistencia al partido elegido. 4. El sistema informa al creador del partido de que el usuario ha cancelado la asistencia.

Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Unirse a partido privado
Actores	Usuario estándar
Descripción	Los usuarios que reciban invitaciones para un partido privado podrán aceptarlas y se unirán al partido creado.
Precondiciones	1. El usuario está registrado y ha iniciado sesión en la aplicación 2. Existe un partido privado 3. El usuario ha sido invitado al partido.
Escenario Principal	1. El usuario accede al menú correspondiente de sus partidos. 2. El usuario acepta la invitación al partido.
Escenario alternativo	No existe escenario alternativo

Nombre	Descripción
Caso de uso	Ocupar partido
Actores	Usuarios complejos deportivos.
Descripción	Los complejos deportivos podrán ocupar lo horarios que deseen de sus campos, eliminando así su disponibilidad.
Precondiciones	1. El complejo deportivo está registrado en la aplicación. 2. El complejo deportivo tiene al menos un campo.
Escenario Principal	1. El complejo accede a sus campos. 2. El complejo accede al calendario del campo que desea ocupar. 3. El complejo reserva la hora que desee.
Escenario alternativo	No existe escenario alternativo

2.2.5 Requisitos no funcionales restantes.

Por último, se van a analizar los últimos requisitos no funcionales del sistema. Se procederá al análisis por separado dado que no son específicos de ninguna iteración, sino que son generales del sistema.

Identificador	Nombre	Descripción
RNF7	Usabilidad	El sistema informará del resultado de todas las acciones que el usuario realice.
RNF8	Interfaz	La aplicación deberá ofrecer una interfaz que permita al usuario realizar las acciones de forma intuitiva.
RNF9	Administración de datos	El sistema interactuará con una base de datos en la que administrará todos los datos de la aplicación.

2.3 Actores

En el lenguaje unificado de modelado con el que se va a realizar el diagrama de casos de uso, un actor representa un usuario o cualquier otro componente que interactúa con el sistema.

En este sistema se encuentran los siguientes actores:

- Usuario normal: Un usuario estándar del sistema, aquel para el que está dedicada la mayor parte de la aplicación.

- Usuario administrador: Es un usuario con los mayores privilegios existente en el sistemas. Estos privilegios le permiten administrarlo.

- Usuario-complejo: Es un usuario diferente al estándar, ya que representa a los complejos deportivos que colaboran con la aplicación. Tiene ciertos privilegios, como poder crear su propio campo o modificarlo, aunque también tiene ciertas limitaciones.

2.4 Diagrama de casos de uso.

Un diagrama de casos de uso modela la funcionalidad del sistema por medio de actores y de casos de uso. Debido a que en este sistema existen varios actores, se ha realizado un diagrama específico para cada actor con el fin de facilitar la comprensión al lector.

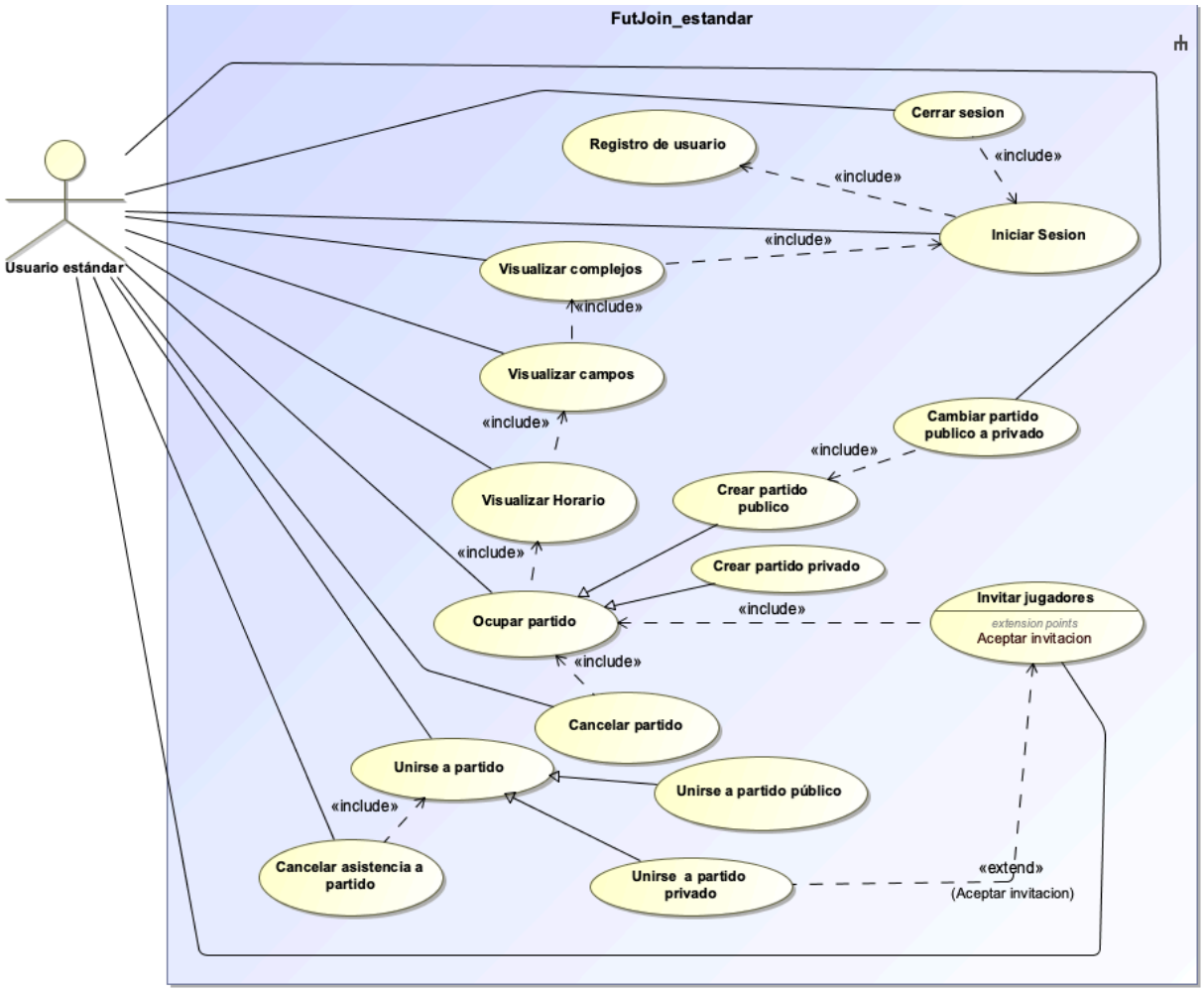


Figura 7. Diagrama de casos de uso del usuario estándar

En la figura 7 se pueden observar los casos de uso del usuario estándar. Lo más destacable es el modelado a la hora de crear partidos e invitar jugadores, en el cual hay que interrelacionar varios casos de uso.

En las siguientes figuras 8 y 9 se encuentran los diagramas correspondientes al complejo deportivo, al usuario administrador y al sistema, que son menos complejos que el del usuario estándar y que incorporan algunos casos de uso especializados para usuarios con sus privilegios.

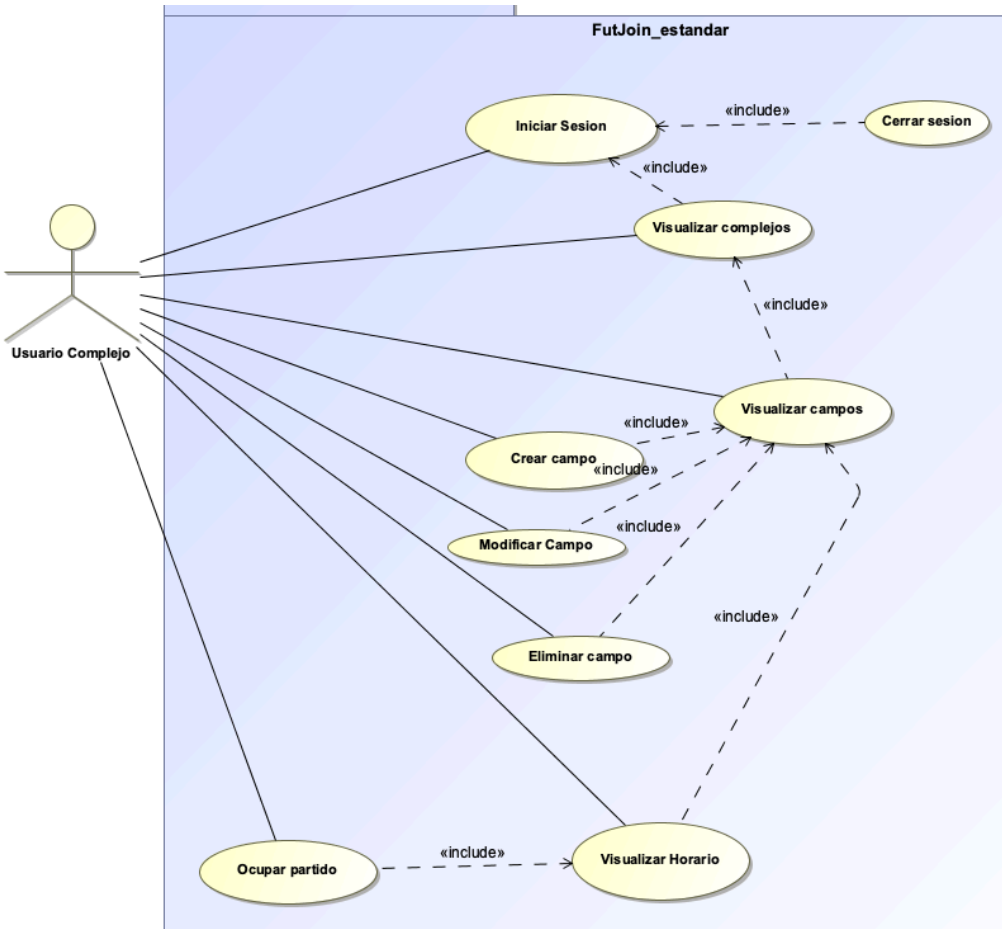


Figura 8. Diagrama de casos de uso del complejo deportivo

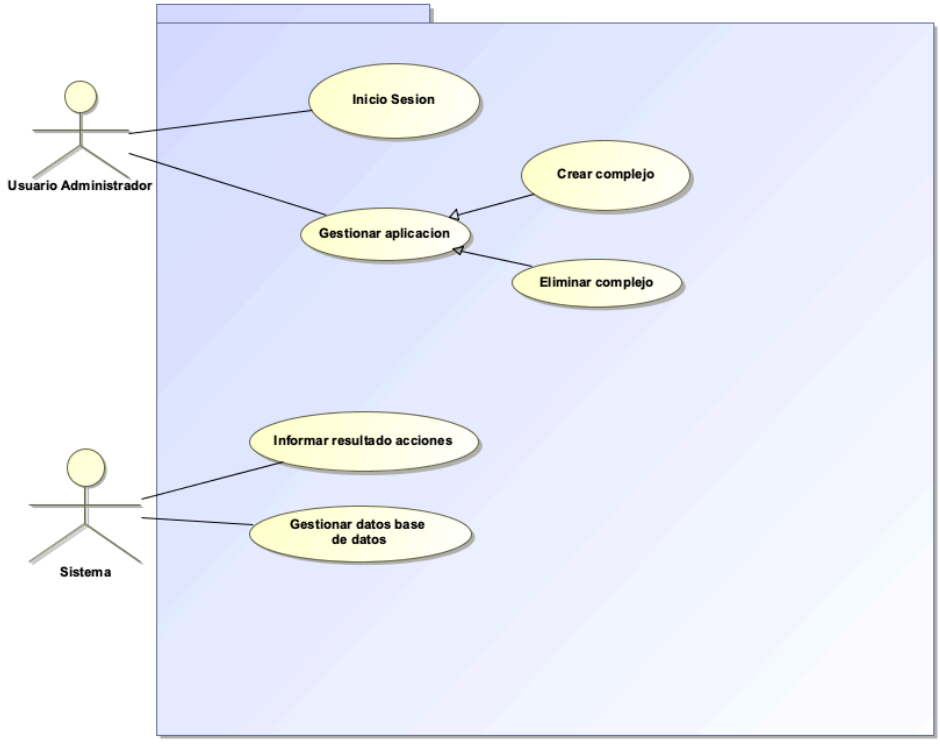


Figura 9. Diagrama de casos de uso del administrador y del sistema

3

Diseño

3.1 Arquitectura Software.

La arquitectura escogida para el desarrollo de la aplicación ha sido la **arquitectura cliente-servidor de tres capas**. Para lograr una mejor comprensión de este concepto, se analizará por partes: en primer lugar, el modelo Cliente/Servidor representa un modelo de diseño en el que las tareas se reparten entre los proveedores de servicios (servidores) y los demandantes, llamados clientes. Las aplicaciones clientes realizan peticiones a los servidores, que atienden dichas demandas.[15] Estos sistemas se denominan arquitecturas de dos capas o niveles. Este tipo de arquitecturas tienen una gran desventaja: el balance de carga es malo, ya que al existir únicamente dos niveles, cada uno de ellos debe realizar demasiadas tareas, sobre todo el servidor, que debe procesar las peticiones de los clientes y analizarlas y, habitualmente, acceder a una base de datos, recoger los datos, procesarlos y devolverlos al cliente. Para solucionar esta sobrecarga surgen las arquitecturas de n-capas. En estas arquitecturas, se añaden capas para repartir la carga de los clientes y servidores. En este caso, se utiliza una arquitectura de tres capas, compuesta por el cliente que interactúa con el usuario final (denominada capa de presentación), el servidor que procesa los datos de los clientes

(denominada capa de negocio) y el servidor de la base de datos que almacena y gestiona los datos que utiliza el servidor (denominada capa de datos).

Las ventajas principales que aportan este tipo de arquitectura son:

- Mayor seguridad, debido a que esta se define de forma independientemente en cada capa.
- Mejor rendimiento, ya que el balance de carga mejora sustancialmente.
- Mayor escalabilidad.

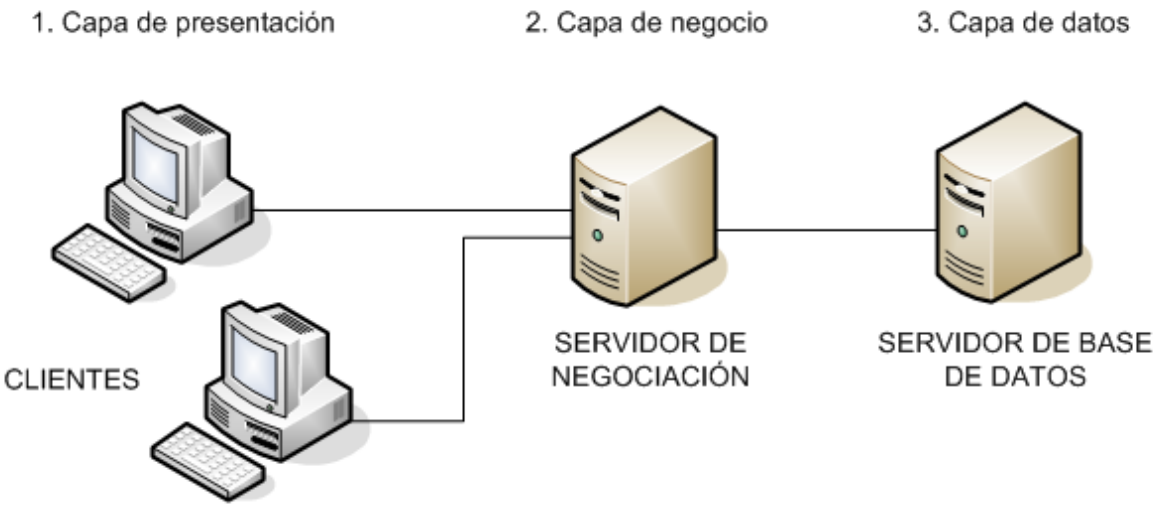


Figura 10. Esquema de arquitectura de 3 capas de una red. Fuente: https://upload.wikimedia.org/wikipedia/commons/e/ea/Tres_capas.PNG

En la figura 10 se muestra el esquema básico de la programación por capas. En el caso de nuestra aplicación, desarrollada con la herramienta web *MEAN Stack* (definida al principio de este trabajo) el funcionamiento es idéntico pero adaptado a la programación web. De esta manera, *Angular* actúa como la capa de presentación, *Node.js* como la capa de negocio y *MongoDB* como la capa de datos que, unido a la ventaja de que todos están basados en el mismo lenguaje de programación, logra un funcionamiento limpio y estable.

3.2 Diseño del servidor (*Back-end*)

Aunque la arquitectura del sistema sea de 3 capas, el diseño y modelado de la base de datos se van a analizar en este apartado dado que, a pesar de que trabajen de forma separada el servidor y la base de datos, están en continuo intercambio de información y e íntimamente relacionados.

3.2.1 API REST

En cuanto al servidor, se ha escogido un estilo de arquitectura tipo REST, generando así una API REST (API: conjunto de funciones y procedimientos que ofrece un sistema y que puede ser utilizado por otro) basada en el protocolo HTTP, que el cliente utilizará para comunicarse con el servidor mediante una capa lógica de usuario. Este protocolo aporta la ventaja de que cualquier aplicación que entienda HTTP podría hacer uso del servidor.

Una vez escogido el protocolo de comunicación, el siguiente paso es decidir qué formato o estructura van a tener los datos que se van a intercambiar entre capas. En este caso, aprovechando que la base de datos trabaja con objetos en formato JSON, este será el formato elegido en el que los datos viajarán a través de los niveles. De esta forma también se consigue homogeneidad en el tipo de mensaje con el que se va a trabajar en todos los niveles.

De esta forma, el cliente se comunica con el servidor mediante los diferentes métodos que el protocolo HTTP aporta para servicios REST: GET, PUT, POST y DELETE.

Cada uno de ellos tiene una funcionalidad específica.

- GET: Método utilizado para recuperar datos. Las peticiones HTTP que utilicen el método GET no deben modificar datos. Como estas peticiones no alteran ningún dato, se califican como métodos seguros.

- POST: Método utilizado para enviar una entidad a un recurso específico. Normalmente causa una modificación o una creación.

- PUT: El método PUT se utiliza principalmente para actualizar o modificar un recurso existente. También pueden crear nuevos objetos. La diferencia con el método POST es que, normalmente, las solicitudes PUT se realizan sobre un recurso individual, mientras que las otras se realizan sobre una colección de ellos.
- DELETE: Como indica el nombre, el método DELETE se utiliza para eliminar un recurso específico.

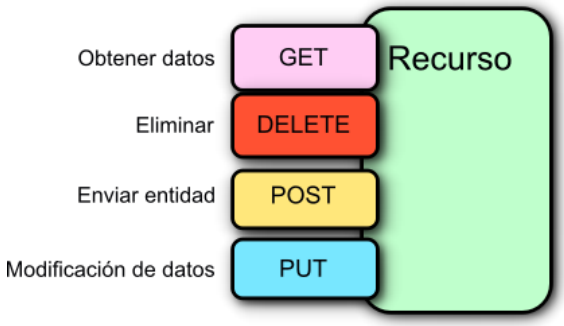


Figura 11. Esquema de métodos HTTP.

Como se podrá observar en la implementación, el diseño del servidor se divide en bloques, relacionados con cada uno de las entidades que componen el sistema. Por lo tanto, encontraremos métodos para los campos, los complejos deportivos, los usuarios y los partidos.

3.2.2 Seguridad

Antes de implementar algoritmos y funciones que otorguen seguridad al servidor, hay determinar el método que se va desarrollar. En este caso, la autenticación de los usuarios es una parte esencial de la aplicación. Para satisfacer esta condición, teniendo en cuenta que nuestro servidor va a alojar una API REST y que estas no manejan estados y por lo tanto no guardan sesiones, se desarrollará un sistema de autenticación por *token*. Un *token* es una cadena de caracteres encriptada única generada a raíz de la validación del usuario (en este caso por correo y contraseña) que se almacena en el cliente, envía al servidor y este utiliza para identificar al usuario. Una de las ventajas de este sistema es que los *tokens* se almacenan en el lado del cliente, por lo tanto este gana una alta escalabilidad y el servidor solo recibe el *token* (en este caso, a través de cabeceras HTTP), lo descifra y redirige el flujo de la aplicación en un sentido u otro. [17] Este sistema elimina las vulnerabilidades web basadas en el *Cross Site Request Forgery* (CSRF) o, en español, la denominada falsificación de petición en sitios cruzados. Para explicar este ataque se va a exponer un sencillo ejemplo:

José es un usuario malicioso que conoce que en un sitio web llamado "futbol-virus.com" existe la opción de que los usuarios puedan enviar mensajes a otro usuario a través de una petición POST que incluya el nombre del usuario y el mensaje a enviar. El usuario, a la vez que está conectado en "futbol-virus.com" accede a su correo electrónico y recibe un email de José formado por un botón que engaña al usuario haciéndole creer que va a ganar dos entradas para un Real Madrid – Barcelona, cuando en realidad este botón es un formulario oculto que realiza esa petición POST a "futbol-virus.com" con los datos de José, de manera que el usuario que ha pulsado el botón ha enviado un mensaje a José sin darse cuenta. Este ejemplo se ha realizado con datos secundarios, sin embargo si en lugar de ser mensajes fuera dinero o datos bancarios, el problema sería mucho mayor.

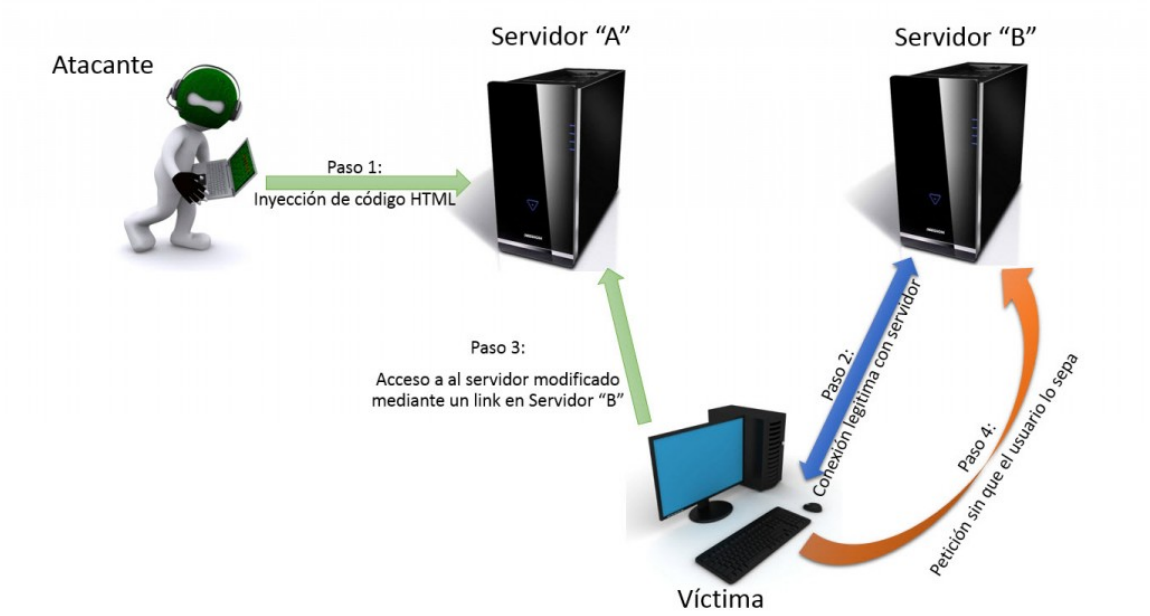


Figura 12. Descripción de CSRF. Fuente: https://www.welivesecurity.com/la-es/2015/04/21/vulnerabilidad-cross-site-request-forgery-csrf/

3.2.3 Base de datos

Una base de datos es un conjunto de información almacenada que pertenece a un mismo contexto ordenada de tal forma que se pueda almacenar y recuperar. Existen las bases de datos relacionales y las no relacionales. Para este proyecto se ha escogido una base de datos no relacional, en concreto una basada en documentos, *MongoDB*.

Para detallar el diseño de la base de datos, va a procederse al análisis del modelo de clases elaborado y de las entidades y relaciones que encontramos en él.

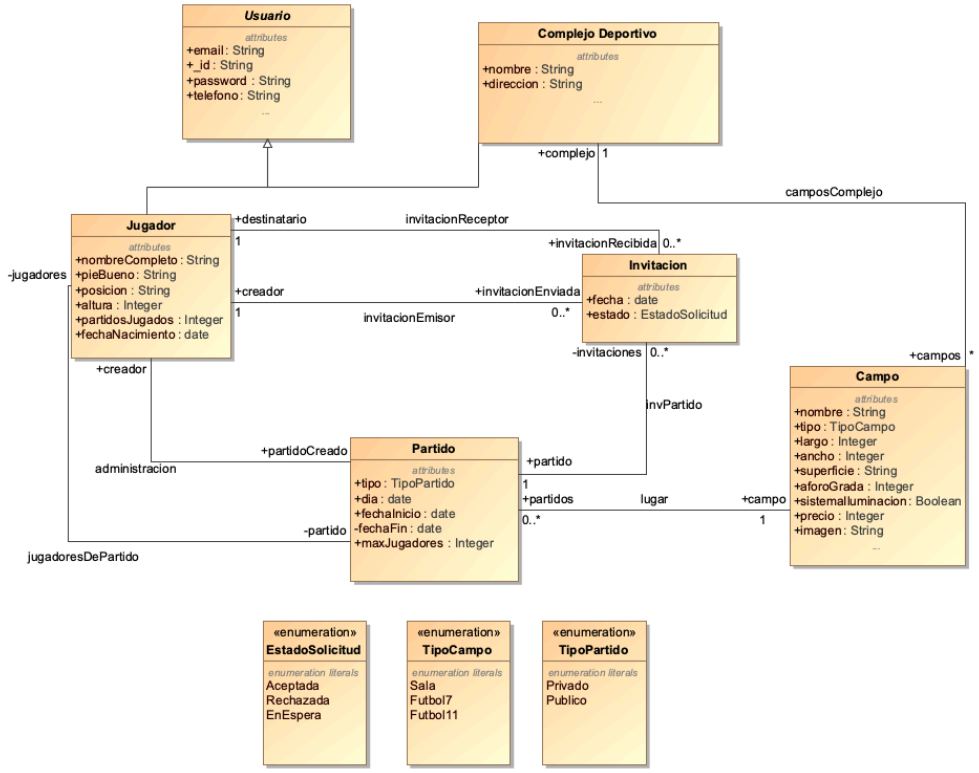


Figura 13. Diagrama de clases de la aplicación.

Las entidades que encontramos en el diagrama son:

- Usuario: Representa el usuario de la aplicación. Es una clase abstracta (una clase que no se puede instanciar) porque un usuario creado siempre va a representar a un jugador o a un complejo deportivo, que son las dos subclases que heredan de ella. En la misma encontramos los atributos generales que van a compartir estas dos: un email, un identificador, una contraseña y un teléfono.

- Complejo Deportivo: Representa uno de los tipos de usuarios que se pueden encontrar. Los atributos que lo componen, además de todos lo del usuario (ya que es una subclase de él) son un nombre y una dirección. El complejo deportivo participa en una relación con la clase Campo, con la que puede acceder a los campos que pertenecen a él.

- Jugador: Representa el otro de los tipos de usuario de la aplicación, el denominado usuario estándar.

Por ello aparecen numerosos atributos relacionados con los datos personales y futbolísticos, como la altura, los partidos jugados o la posición que ocupa en el campo. También se puede apreciar que es la que más relaciones mantiene con el resto de las clases, ya q que va a interactuar continuadamente con la aplicación.

- Campo: Es la clase que representa un campo de fútbol perteneciente a un determinado complejo deportivo. Contiene el mayor número de atributos de todo el modelo. Esto manifiesta su gran importancia, ya que es el lugar donde se va a jugar el partido. Por lo tanto, el usuario necesita conocer adecuadamente sus características. Como consecuencia, encontramos: las medidas del campo, representadas con el largo y el ancho, el tipo de superficie que lo compone, el tipo de fútbol que se puede practicar en él (mediante el enumerado *TipoCampo*), si cuenta con sistema de iluminación o no (mediante un objeto de tipo booleano), el precio que cuesta reservarlo, el aforo de la grada (si la tiene) y una imagen para que el usuario pueda conocer visualmente el campo . El calendario se ha modelado como relación con la clase partido, en la que el campo puede consultar los partidos organizados en él, así como horarios y a raíz de ello crear una agenda en la interfaz.

- Partido: Representa un partido organizado a través de la aplicación. En él se puede encontrar los datos relacionados con el horario en el que se va a jugar, como el día, la hora de inicio del partido y la hora final. También maneja una gran relación con los usuarios jugadores, ya que cuenta con un atributo con el máximo de jugadores que pueden jugarlo (que dependerá del tipo del campo en el que se practique) además de dos relaciones con el usuario jugador que representan los jugadores que van participar en el partido y el jugador que se va a encargar de crearlo y por lo tanto, gestionarlo.

- Invitación: Esta clase se introdujo en el diagrama debido a la aplicación de un proceso de reificación entre la clase Jugador y Partido. Al tener la obligación de controlar un conjunto de datos al crear un partido, se decidió crear una clase intermedia llama Invitación en la que un jugador envía una invitación de un partido a otro jugador, que actúa como receptor. Por ello, contiene dos relaciones con la clase jugador y una con la clase partido.

Además, esta clase contiene un atributo con la fecha y otro con el estado de la solicitud a través de un enumerador denominado *EstadoSolicitud*.

- Enumeradores: Un enumerador es un tipo de dato software que lista un conjunto de valores o elementos llamados enumerados. En este diagrama podemos encontrar tres. En primer lugar, *EstadoSolicitud*, el cual representa el estado de una invitación, que puede ser aceptada, rechazada o estar en espera. En segundo lugar, *TipoCampo* hace alusión a la clase de fútbol que se va a practicar, pudiendo ser fútbol sala, fútbol 7 o fútbol 11. Por último y en tercero, *TipoPartido* define si un partido creado es público o privado.

3.3 Diseño del cliente (*Front-end*)

El diseño del cliente viene prácticamente limitado por el *framework* que se ha elegido para trabajar. Una de las peculiaridades de *Angular* es que se organiza en componentes. Un componente sólo es un pequeño fragmento del sistema encapsulado y totalmente reutilizable. Por lo tanto, podemos corroborar que Angular utiliza un patrón de diseño *composite* o compuesto, que permite diseñar sistemas complejos partiendo de otros mucho más simples. Dentro de estos componentes, se recurre al patrón MVVM.

El patrón MVVM (*Model-View-View-Model*) en el que el modelo y la vista son totalmente dependientes, surge de un refinamiento del clásico patrón *Modelo-Vista-Controlador* (MVC). Esto se debe a que *Angular* emplea una metodología llamada enlace bidireccional, que encaja a la perfección con él. En este patrón el controlador es sustituido por lo que se denomina el *ModeloVista* o *ModelView*, de manera que si se actualizan datos en la vista, inmediatamente se van a actualizar en el modelo, sin necesidad de la existencia de un controlador intermedio. Aunque el concepto es muy parecido al del MVC, se adapta mucho mejor a la funcionalidad de un cliente web.

Como resultado de la entrevista hecha para el análisis de requisitos, también se pueden tomar decisiones sobre el diseño de la interfaz del cliente. En primer lugar, dado que se va a dirigir a un público joven, se decidió elegir como colores principales de la aplicación el azul y el verde; el verde es el color característico del fútbol, ya que se relaciona con el césped del campo y el azul se asocia a la energía física y al deporte.

A raíz de esta decisión se diseñó el logo de la aplicación, el cual se compone de los dos colores elegidos junto a la silueta de un jugador de fútbol con el fin de llamar la atención de los amantes de este deporte.

Figura 14. Logo de la aplicación. Diseño: Javier Boned.

4

Implementación y pruebas

La fase de implementación representa la realización del código de la aplicación, junto a las pruebas realizadas para comprobar su correcto funcionamiento al ejecutarlo.

Antes de comenzar a analizar la implementación del servidor y cliente, se debe añadir que durante el desarrollo de la aplicación se ha creado una nueva clase para facilitar información al usuario: las noticias. Las noticias representan sucesos de la aplicación que tienen suficientemente relevantes como para informar a todo usuario que inicie sesión en ella. En las sucesivas partes de este capítulo se explicará cuando y por qué se hace uso de ellas. También es de aclarar que tanto en el servidor como en el cliente se va a mostrar una carpeta llamada *node_modules*. Esta carpeta contiene todas las librerías descargadas a través de el gestor de descargas de *Node.js*, utilizado en ambas implementaciones. Estas librerías se analizarán por separado tanto en cliente como en servidor.

4.1. Implementación del servidor.

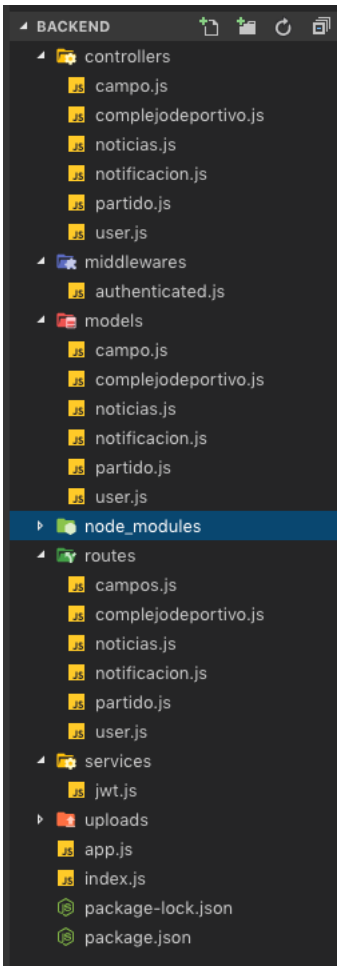


Figura 15. Estructura del servidor.

Como se puede observar en la figura 15, el servidor *Node.js* se organiza en 3 grandes carpetas: "*controllers*", donde se encuentran las funciones de la API que van a acceder a la base de datos, "*models*" que representan los modelos de las entidades con todos sus respectivos atributos y "*routes*", que establece las rutas HTTP por las que el cliente va a comunicarse con el servidor. Todas ellas se estructuran conforme a las iteraciones del sistema: usuario, complejo deportivo, campo, invitación (notificación) y noticias.
Además, hay otros archivos exclusivos del lenguaje, como "*app.js*" que es el archivo principal de un directorio *Node.js*.
También encontramos otras carpetas como la de servicios o la de middlewares, en la que se implementa el estándar JWT (*Json Web Tokens*), el cual se analizará en profundidad mas adelante y que permite la autenticación de los usuarios a través de tokens.

4.1.1 JWT

Antes de analizar las funciones creadas en la API REST, se va a proceder a la descripción del estándar utilizado para la creación de *tokens* de acceso. JWT trabaja en formato JSON, por lo que resulta inmejorable para una aplicación MEAN. El funcionamiento de JWT es el siguiente:

El *token* encriptado generado en JWT está compuesto por 3 cadenas separadas por un punto entre ellas. La primera es la cabecera (también llamada *header*), que contiene el tipo (en este caso JWT) y la codificación utilizada.
La segunda engloba los atributos que definen el *token*, entre los que destacan "*sub*", que es el encargado de identificar al usuario, "*iat*" que representa la fecha de creación

del *token* y "*exp*", que concreta la fecha de expiración de él. La tercera y última parte se denomina firma (o *signature*) y está formada por las dos anteriores cadenas cifradas en Base64 con una clave secreta que almacenaremos en el servidor. A continuación se muestra un *token* ejemplo con las tres partes diferenciadas por colores.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1ZDA2NjQyZGQ2MWNmNDAzZjcyZmM5ODgiLCJuYW1lIjoiSmF2aWVyIEJvbmVkIExvcGV6IiwiZW1haWwiOiJqYXZpYm9uZWRAZ21haWwuY29tIiwiaW1hZ2UiOiJHX0Y1djR2eTZ5c1BmamlOVzlMTUFuRlEucG5nIiwicm9sZSI6MSwiaWF0IjoxNTYxMDc3ODAyfQ.NVXv0zv7gd1YfqeT9PiFFR2trf0b0-G7KquJjDg8iJY

4.1.2 Funciones de la API REST

A continuación vamos a analizar los métodos de nuestra API, divididos por entidades del sistema.

- Usuarios.
  - RegistrarUsuario: Método POST que recibe como parámetros en el cuerpo de la petición HTTP todos los datos que un usuario introduce al registrarse en la aplicación. Con ellos crea un objeto de tipo usuario y lo introduce en la base de datos.
  - Login: Método POST que recibe como parámetros un email y una contraseña y comprueba si el usuario existe en la base de datos. Este método utiliza la librería JWT para crear un token de autenticación.
  - ActualizarUsuario: Método PUT en el que un usuario envía como parámetros los campos que quiere modificar.
  - SubirImagen/obtenerImagen: El objetivo de estos dos métodos es administrar las imágenes de perfil de cada usuario. Permiten que podemos guardarlas en el servidor y generar un enlace para que el cliente pueda visualizarlas.
- ComplejoDeportivo

- guardarComplejo: Método POST que recibe como parámetros los datos de un campo y lo inserta en la base de datos.

- getComplejos: Método GET para obtener la lista de complejos deportivos.

- deleteComplejo: Método DELETE para eliminar un complejo. Este método tiene un poco más de complejidad que el resto a la hora de implementarlo dado que hay que eliminar todas las entidades que dependan de un complejo deportivo (denominado borrado en cascada).

❖ Campos

- obtenerCampos: Método GET para obtener la lista de campos de un complejo deportivo. Recibe como parámetro el identificador del complejo deportivo y el tipo, ya que los campos se mostrarán organizados por el tipo de fútbol que se puede practicar en ellos.

- guardarCampo: Método POST que recibe como parámetros los datos de un campo y lo inserta en la base de datos.

- actualizarCampo: Método PUT que modifica un campo.

- subirImagenCampo / ObtenerImagenCampo : Métodos que tienen exactamente la misma funcionalidad que los del usuario, pero con la foto del campo.

- eliminarCampo: Método DELETE para eliminar un campo.

❖ Notificación

- obtenerNotificacionesRecibidas: Método GET para obtener las notificaciones recibidas de un usuario.

- guardarNotificaciones: Método POST para crear invitaciones que crea el usuario emisor.

- eliminarNotificaciones: Método DELETE para eliminar una invitación.

4.1.3 Librerías utilizadas.

Todas las librerías utilizadas se han instalado a través de el gestor de paquetes de Node.js: npm.

i. Bcrypt-nodejs: Librería para encriptar las contraseñas a través de funciones hash.

ii. Body-parser: Middleware para analizar de forma sencilla los cuerpos de solicitudes entrantes al servidor. Proporciona módulos para analizar cuerpos en formato JSON.

iii. Connect-multiparty: Middleware para facilitar el desarrollo de las rutas de nuestra API.

iv. Moment: Librería para manipular fechas en JavaScript.

v. Mongoose: Herramienta de modelado de objetos MongoDB diseñada para trabajar en entornos asíncronos. [18]

4.2 Implementación del cliente.

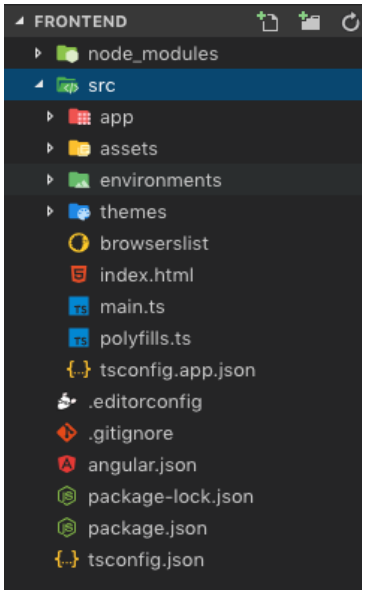


Figura 16. Estructura cliente Angular.

En cuanto al cliente, al implementarlo en *Angular*, encontramos numerosos archivos de configuración que el framework crea automáticamente al iniciar un proyecto. Estos archivos engloban un cúmulo de parámetros relacionados con el funcionamiento del sistema: desde las librerías utilizadas hasta importaciones de *scripts*. En este conjunto podríamos incluir la carpeta "environmets", y los archivos "*angular.json*", "*package-lock.json*", "*package.json*", "*tsconfig.json*", "*browserslist*", "*polyfills.ts*" y "*ts.config.app.json*".

Además de los archivos de configuración, la estructura contiene la carpeta "themes" donde se encuentra los temas y estilos utilizados por la aplicación. En esta carpeta se definen los colores principales de la aplicación y los diferentes elementos de estilo que se van a utilizar durante todo el desarrollo, como la tipografía.

Sin embargo, la carpeta más importante del cliente y sobre la que el programador trabaja constantemente, es "app". Ella contiene los componentes principales de la aplicación.

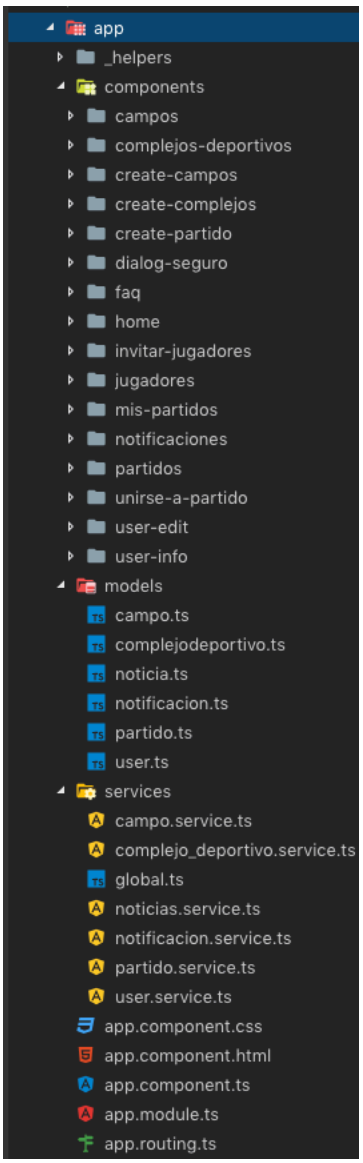


Figura 17. Contenido carpeta "App".

En la figura 17 se presenta el contenido de la carpeta "app". La carpeta "*components*" contiene los componentes que van a conformar el sistema, "*models*" los modelos de las entidades junto con sus atributos y "*services*" los servicios que van a contactar con la API del servidor.

4.2.1 Componentes

Los componentes son la base de la estructura de una aplicación implementada en *Angular*. Cada uno de ellos cumple una función y son totalmente reutilizables. Esto provoca una gran escalabilidad. Cada componente está formado por un archivo con extensión *html* para crear el documento web, otro con extensión *CSS* para especificar la presentación de los elementos en él y un archivo con formato *TS* (*TypeScript*) donde se definen las propiedades, las funciones y los módulos del componente. Este archivo es el que interactúa con el servidor y organiza el funcionamiento web.

Se va a proceder al análisis de todos los componentes que construyen la aplicación.

- Componente Campos: Se encarga de mostrar una tabla expansible con los campos de un determinado complejo deportivo.

- Componente Complejos-Deportivos: Muestra imágenes e información básica de los complejos deportivos que colaboran con la aplicación a través de una animación denominada carrusel.

- Componente Create-Campos: Crea un campo. Solo puede acceder a él el complejo deportivo que quiera crear el campo, por lo tanto no es un componente visible para los usuarios estándar.

- Componente Create-Complejos: Crea un complejo deportivo. Solo puede acceder a él el usuario administrador del sistema.

- Componente Dialog-Seguro: Componente cuyo propósito es crear una ventana de confirmación consultando al usuario si está seguro de la acción que va a realizar.

- Componente FAQ: Encargado de crear la lista de preguntas frecuentes.

- Componente Home: Componente del inicio de la aplicación

- Componente Invitar-jugadores: Ofrece al usuario creador de un partido la posibilidad de invitar a otros jugadores, mostrándole una lista con los perfiles de todos los usuarios registrados en la aplicación junto a un buscador para filtrarlos según el usuario desee.

- Componente Jugadores: Muestra la lista de jugadores de un partido a través de (nuevamente) un carrusel con sus imágenes de perfil.

- Componente Mis-Partidos: Componente para consultar los partidos de un usuario, tanto de los que es creador como de los que es participante.

- Componente Notificaciones: Permite visualizar las invitaciones de partidos que ha recibido el usuario, así como las confirmaciones y cancelaciones que ha enviado.

- Componente Partidos: Calendario de un determinado campo, donde se muestran tanto las horas libres como las ocupadas por los partidos creados.

- Componente Unirse-a-partido: Encargado de ofrecer la posibilidad al usuario de unirse a un partido.

- Componente User-edit: Componente que permite a los usuarios visualizar su perfil y modificar sus datos.

- Componente User-info: Perfil público de cada jugador.

- Componente App-Component: Componente raíz del sistema. Se ejecuta en el inicio de la aplicación y a partir de él, el usuario puede acceder al resto de componentes. En este caso, representa la pantalla inicial, donde el usuario puede realizar el *login* o registrarse en la aplicación.

4.2.2 Servicios y Modelos

Los servicios son las funciones de *Angular* que se encargan de generar las peticiones HTTP y enviarlas a la API desarrollada en el servidor. Estas peticiones pueden ir acompañadas de parámetros que incluyan información o no. Los servicios se dividen también conforme a las entidades de la aplicación, de modo que cada iteración tendrá su propio archivo de servicios, representado en *Angular* como <nombreDeEntidad>.*service.ts.*

En cuanto a los modelos, su implementación es prácticamente igual que en el servidor, con la diferencia de que está escrito en el lenguaje *TypeScript* (muy similar a *JavaScript*) añadiendo algunos parámetros específicos de este.

4.2.3 Librerías utilizadas

Todas las librerías utilizadas se han instalado a través del gestor de paquetes *npm.*

- Angular Material: Librería similar a *Bootstrap* creada por *Google* con el fin de mejorar el diseño de aplicaciones web desarrolladas en Angular. Contiene numerosos componentes para implementar una interfaz moderna, atractiva y consistente. Algunos de ellos son: botones, formularios, tablas, barras de progreso, iconos, etc…

- Ngx-carousel-3d: Librería para crear carruseles en Angular. Un carrusel es una galería de imágenes animada.

- Ngx-float-button: Librería para diseñar botones especiales con funcionalidades avanzadas.

- Ngx-toastr: Librería para poder crear componentes de alerta (cuya duración son pocos segundos) que notifiquen al usuario del estado del sistema.

4.3. Pruebas.

Para localizar los errores que pudiesen surgir y solucionarlos, se recurrió a 5 usuarios que probaran la funcionalidad de la aplicación. Estos usuarios se eligieron minuciosamente, ya que tenían que pertenecer al público objetivo de la aplicación. Los errores, con su descripción y la solución desarrollada, son los siguientes:

- Sesión no persistente: La información de la sesión del usuario no persistía y al actualizar la página se perdía automáticamente. La solución desarrollada fue recurrir al *LocalStorage*, herramienta de *JavaScript* para sustituir a las sesiones.

- Posición portero: Cuando el usuario realizaba el registro, a la hora de introducir los datos futbolísticos, no se contemplaba la opción de que el sujeto no fuera jugador de campo. La solución fue añadir la opción de "portero" en la lista de posiciones.

- Dimensiones del elemento de registro: Si el usuario empequeñecía un poco la pantalla, el formulario de registro quedaba totalmente desorganizado. Se tuvo que implementar CSS especial para este elemento para ajustarlo.

- Foto de perfil de usuario no se adapta corrientemente: Con el fin de que la foto de usuario se adaptara a todas las pantallas, se implementó código CSS de manera incorrecta y los usuarios con pantallas muy grandes veían la foto de perfil con tamaño excesivo. Por lo tanto, se corrigió estableciendo fijos los parámetros de altura y de anchura de una foto.

- Horario no tenía todas las horas: El horario de un campo de fútbol comprende todas las horas desde las 9 de la mañana hasta las 11 de la noche. El usuario se

detectó que el horario carecía de una hora en el calendario. Se añadió sin problema y se solucionó la cuestión.

- Problema al modificar objetos dinámicamente: El usuario manifestó el problema de que cuando eliminaba un partido, convertía un partido de público a privado o cancelaba su asistencia a un partido, los elementos no se eliminaban dinámicamente de la pantalla, situación que le creaba confusión. Para esto, por medio de herramientas de *TypeScript*, se convirtieron todas las listas y tablas a elementos dinámicos, de forma que la interfaz plasme la respuesta de la acción del usuario de forma inmediata. Esto crea una sensación de fluidez que complace al usuario.

Debido a que las funcionalidades desarrolladas para el usuario administrador y para el que representa un complejo deportivo son escasas y básicas, han sido repetidamente probadas por el mismo programador, a través de diversas herramientas.

5

Conclusiones

5.1 Conclusiones

Este trabajo de fin de grado tenía un objetivo claro: aprender a dominar un lenguaje que se encuentra absolutamente en alza en los últimos tiempos en el entorno de la programación web. Este lenguaje es *JavaScript*.

Una vez seleccionado el lenguaje que iba a emplear y los propósitos académicos que quería satisfacer, debía decidir la temática sobre la que iba a desarrollar la aplicación. Después de barajar varias posibilidades que no resultaron concluyentes, una experiencia personal me ayudó a decantarme por la temática de la aplicación. Como gran aficionado al fútbol, a menudo me encontraba con la situación de organizar partidos de fútbol entre amigos y contantemente echaba en falta una plataforma que utilizáramos todos que nos facilitara la organización de estos partidos. Situación que no era fácil de lograr por diversos motivos: ausencia de jugadores, dificultad de comunicación con los complejos deportivos, etc…

Además, el hecho de trabajar con datos relacionados con el mundo del fútbol (que me son tremendamente familiares) suponía una ventaja añadida para el desarrollo del proyecto. Me pareció que, al ser un problema real que surgía en la vida cotidiana y al ser una temática que me ha apasionado a lo largo de mi vida, el resultado podía ser magnífico.

Sin embargo, no resultó una tarea fácil. En la universidad había adquirido conocimientos de desarrollo web a través de diferentes lenguajes de programación como *Python* o *Java* que me ayudaron para diseñar la aplicación y tuve que realizar un estudio de las técnicas que requiere la utilización de *JavaScript*. Durante este estudio encontré la herramienta web *MEAN Stack*, que parecía idónea para realizar una aplicación con un servidor y un cliente de forma eficaz. Tecnologías que encajaban de forma ideal entre ellas y que utilizaban el lenguaje elegido, junto al hecho de utilizar bases de datos no relacionales, con las cuales nunca había trabajado y que me despertaban una gran curiosidad.

Esto supuso que las fases de análisis y diseño fueran mucho más sencillas de realizar que la de implementación, ya que son más generales y se enseñan de forma extensa durante el grado. En cuanto a la implementación, la realización de la API del servidor no generó problemas de gran importancia, pues conocía el funcionamiento de ellas y una vez aprendido el lenguaje, no fue difícil adaptar estos conocimientos al desarrollo.

El inconveniente principal lo encontré en la implementación del cliente. En primer lugar, *Angular* es un *framework* cómodo, pero en mi modesta opinión difícil de entender. No acostumbrado a trabajar con componentes ni con clientes web, el principio me resultó un poco farragoso. Este es el motivo de la creación de alguna entidad más durante la implementación, como las noticias, o que el resultado final difiera un poco de lo que originariamente se había plantead. Esto me ha ayudado a comprender que a lo largo del desarrollo de una nueva aplicación con una tecnología poco conocida nos podemos encontrar inconvenientes sobrevenidos que hay que ir resolviendo. Sin embargo, acudiendo a la documentación y bibliografía, pude solventar las dudas que me fueron surgiendo a lo largo del proceso. Otro elemento con el que no estaba familiarizado y que ha supuesto un esfuerzo añadido es el diseño web. Debido a la creatividad para el diseño visual de la que adolezco, todas las tareas relacionadas con

este han supuesto un inconveniente añadido durante todo el desarrollo: elección de colores, diseño de botones, colocación de elementos, diseño *responsive*, etc…

A pesar de los contratiempos surgidos, pienso que el resultado final de la aplicación ha cumplido mis expectativas. Visualmente es agradable para el usuario, implementa numerosas animaciones que dan una sensación de fluidez constante y la funcionalidad se cumple a la perfección, pudiendo organizar partidos en cuestión de minutos. Me encuentro muy satisfecho con el resultado obtenido y con todo lo aprendido durante el desarrollo del sistema. La realización de este trabajo me ha permitido tomar consciencia de que todos los conocimientos aprendidos durante estos años tienen una aplicación práctica y tangible.

5.2 Líneas futuras.

Pienso que la aplicación tiene un gran margen de mejora. En este momento, la funcionalidad de la aplicación satisface los objetivos básicos: registro e inicio de sesión de un usuario, colaboración de complejos deportivos junto a sus campos y creación de partido junto a la posibilidad de invitar a otros jugadores registrados en la aplicación. Una vez cimentada esta base, sobre esta se pueden desarrollar múltiples funcionalidades.

Por ejemplo, se podría ofrecer un sistema de puntuación para cada usuario, de forma que unos jugadores puedan puntuar a otros dependiendo del nivel de fútbol que tengan, Esto generaría competitividad sana en la aplicación y le aportaría un componente social. También se podrían publicar los resultados de los partidos, eligiendo al mejor jugador del partido o introducir los goles marcados por cada jugador, de manera que cada vez se tenga más información de los datos futbolísticos de los usuarios. Esto quizás generaría la atención del mundo del fútbol, aunque fuese a niveles aficionados.

Además, aunque estoy satisfecho con la interfaz, la capacidad de mejora de esta es evidente. Mejorar los iconos, mejorar el diseño *responsive* o colocar los elementos en mejor disposición serían algunas de las posibilidades existentes para mejorar la interfaz del sistema, que está realizada por alguien con poco conocimiento en este ámbito. Además, *Angular* ofrece múltiples posibilidades para mejorar el entorno visual y con unos conocimientos profesionales se podría conseguir un resultado muy atractivo.

Sin embargo, la aplicación tiene como objetivo primordial facilitar la reserva de campos y creación de partidos para personas que realicen el deporte de forma amateur, por lo tanto creo que añadir demasiadas funcionalidades la convertirían en un sistema más profesional y podría perder el público para el que realmente está destinada. Descuidar el objetivo principal de la aplicación supondría un error grave.

Bibliografía

[1] Subdirección General de Estadística y Estudios, Secretaría General Técnica Ministerio de Educación, Cultura y Deporte. (2015). Encuesta De Hábitos Deportivos 2015. Sintesis De Resultados ahora en Calameo., p.7.
Disponible en: http://www.culturaydeporte.gob.es/servicios-al-ciudadano-mecd/dms/mecd/servicios-al-ciudadano-mecd/estadisticas/deporte/ehd/Encuesta_de_Habitos_Deportivos_2015_Sintesis_de_Resultados.pdf [Accesible 5 Feb. 2019].

[2] FIFA. FIFA big count 2006: 270 million people active in football. (2017). Centro Studi Federale – Rome: FIFA. Available at: https://es.fifa.com/mm/document/fifafacts/bcoffsurv/smaga_9472.pdf [Accesible 5 Feb. 2019].

[3] Pedro Santamaría. Timpik, la aplicación social para deportistas.(18 de marzo de 2013.) Disponible en: https://www.applesfera.com/aplicaciones-ios-1/timpik-la-aplicacion-social-para-deportistas

[4] Biwenger, el mánager de fútbol fantasy oficial de AS y Cadena SER. https://www.biwenger.com/

[5] Flanagan, D. (2006). JavaScript: the definitive guide. “O'Reilly Media, Inc.” Pagina 1. Disponible en: Link del libro.

[6] Bierman, G., Abadi, M., & Torgersen, M. (2014, July). Understanding typescript. In European Conference on Object-Oriented Programming.Springer, Berlin, Heidelberg. Disponible en: https://link.springer.com/chapter/10.1007/978-3-662-44202-9_11

[7] Haviv, A. Q. (2014). MEAN Web Development. Packt Publishing Ltd. Chapter 1. Introduction to Mean. Disponible en: Link del libro

[8] Angular (framework) - Wikipedia, la enciclopedia libre.
https://es.wikipedia.org/wiki/Angular_(framework)

[9] Fundacion Node.js. (2018). Acerca | Node.js. Página oficial.
https://nodejs.org/es/about/

[10] Fundacion Node.js. (2018). Express - Node.js web application framework. Pagina Oficial. https://expressjs.com/es/

[11]MongoDB - Wikipedia, la enciclopedia libre
https://es.wikipedia.org/wiki/MongoDB

[12] David Vicente. (9 de marzo de 2017). Encriptación de password en NodeJS y MongoDB:bcrypt.https://solidgeargroup.com/password-nodejs-mongodb-bcrypt?lang=es

[13] Desarrollo iterativo y creciente - Wikipedia, la enciclopedia libre.
https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[14] https://www.uml-diagrams.org/use-case-include.html

[15] Marini, E. (2012). El Modelo Cliente/Servidor. Recuperado el, 5.

[16] Mozilla y colabadores individuales.(2005-2019). Métodos de petición HTTP - HTTP | MDN. https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[17] Carlos Azaustre (Febrero 2015). Qué es la autenticación basada en Token. https://carlosazaustre.es/que-es-la-autenticacion-con-token/

[18] Npm, Inc. mangosta – npm. https://www.npmjs.com/package/mongoose

Apéndice A

Manual de Usuario

1. Pantalla de inicio

En la pantalla de inicio encontraremos el logo de la aplicación y el formulario de inicio de sesión, junto a las opciones de regsitrarse en la aplicación y de contactar con la aplicación en caso de que el usuario sea un complejo deportivo.

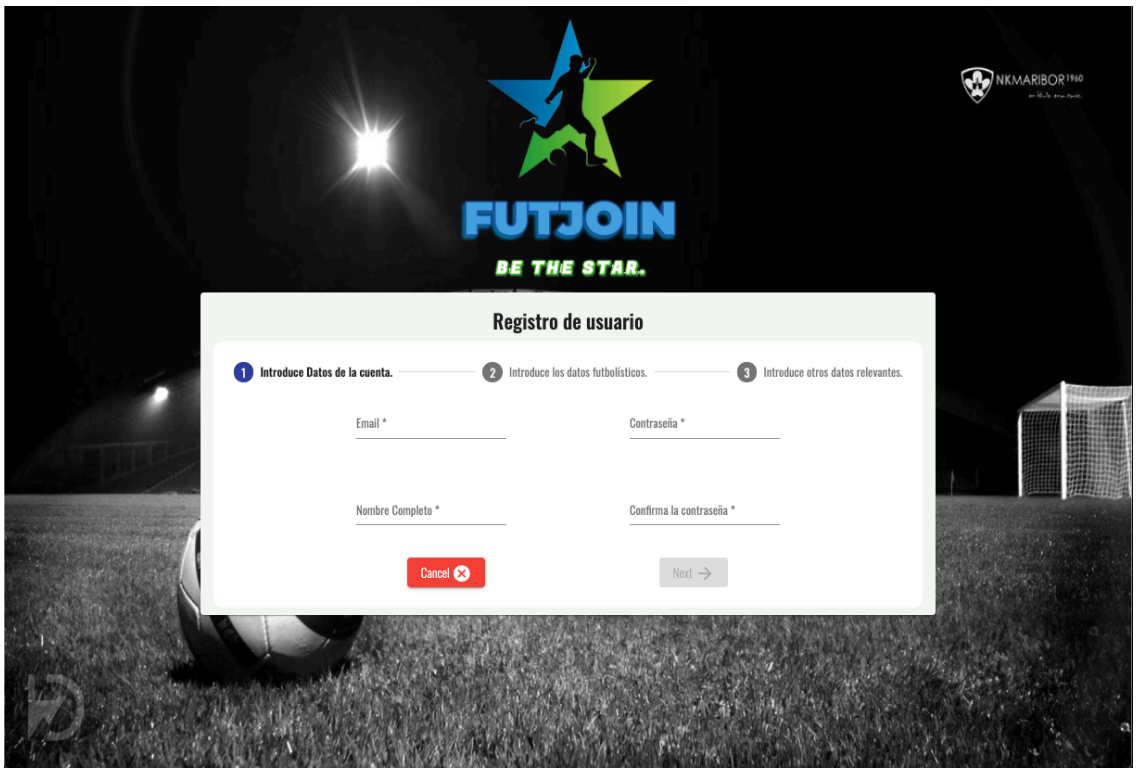
1.1 Inicio de sesión

1. Introducirsu correo en el apartado de “email”.
2. Introducir su contraseña en el apartado de “contraseña”.

3. Pulsar Iniciar sesión

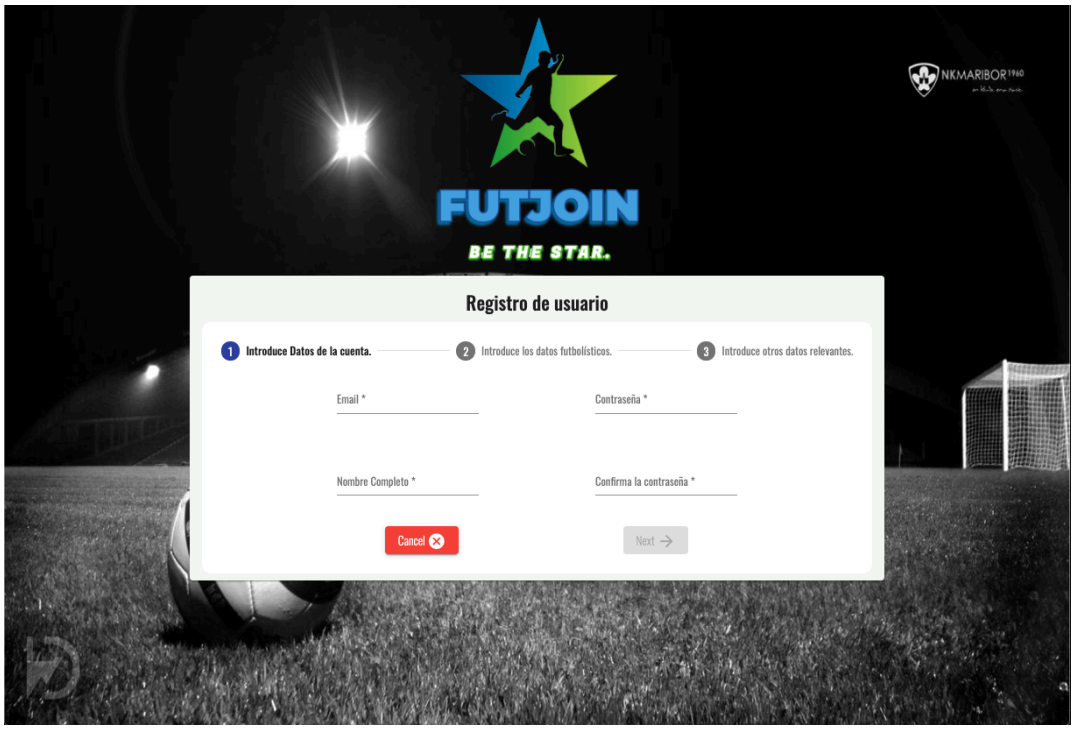
1.2 Registro de usuario

1. Pulsar el botón “¿No tienes cuenta? Registrate.”
2. Seguir las instrucciones del formulario por partes que aparece en la pantalla.
3. Pulsar en “Finalizar”.



1.3 Contacto de complejo

1. Pulsar el botón “¿Eres complejo deportivo? ¡Contáctanos!.”
2. Rellenar los datos del formulario que aperece en pantalla.
3. Pulsar en “Enviar”.



2. Home

En el “home” de la aplicación se encuentra una cabecera y una lista de las noticias de la aplicación. Encontrará dos barras principales, una horizontal situada en la parte superior y una vertical presentada a la izquieda.

2.1 Barra vertical izquierda

La barra vertical izquierda es el menú principal de la aplicación. En ella encontramos los menús principales. En primer lugar, con un icono de una casa, se encuentra el home ya explicado. En segundo lugar, con una pesa, encontramos representados los complejos deportivos. En tercer lugar, con una pelota, los partidos del usuario logueado y en cuarto lugar, con una campana, las notificaciones recibidas así como las enviadas no leídas, junto al número de notificaciones encima de la campana. Al final de la barra hay otros dos menús: uno con una persona representando los datos del usuario iniciado y uno con el símbolo de apagar donde se puede cerrar la sesión.
Además, esta barra puede ensancharse pulsando el botón de las tres rayas horizontales situado encima.

2.2 Barra superior

Esta barra tiene menos importancia, aunque muestra el logo de aplicación y a la derecha contiene un botón con 3 puntos verticales donde se puede acceder al FAQ y podemos cerrar la sesión

3. Complejos deportivos

1. Pulsar el botón de la pesa de la barra vertical izquierda.

2. Pular las flechas rosas para avanzar o retroceder en el carrusel que muestra la imagen del complejo y sus datos.

3. Pulsar el botón “Fútbol 7”, “Fútbol 11” o “Fútbol Sala” para visualizar los campos de ese tipo que pertenecen al complejo deportivo.

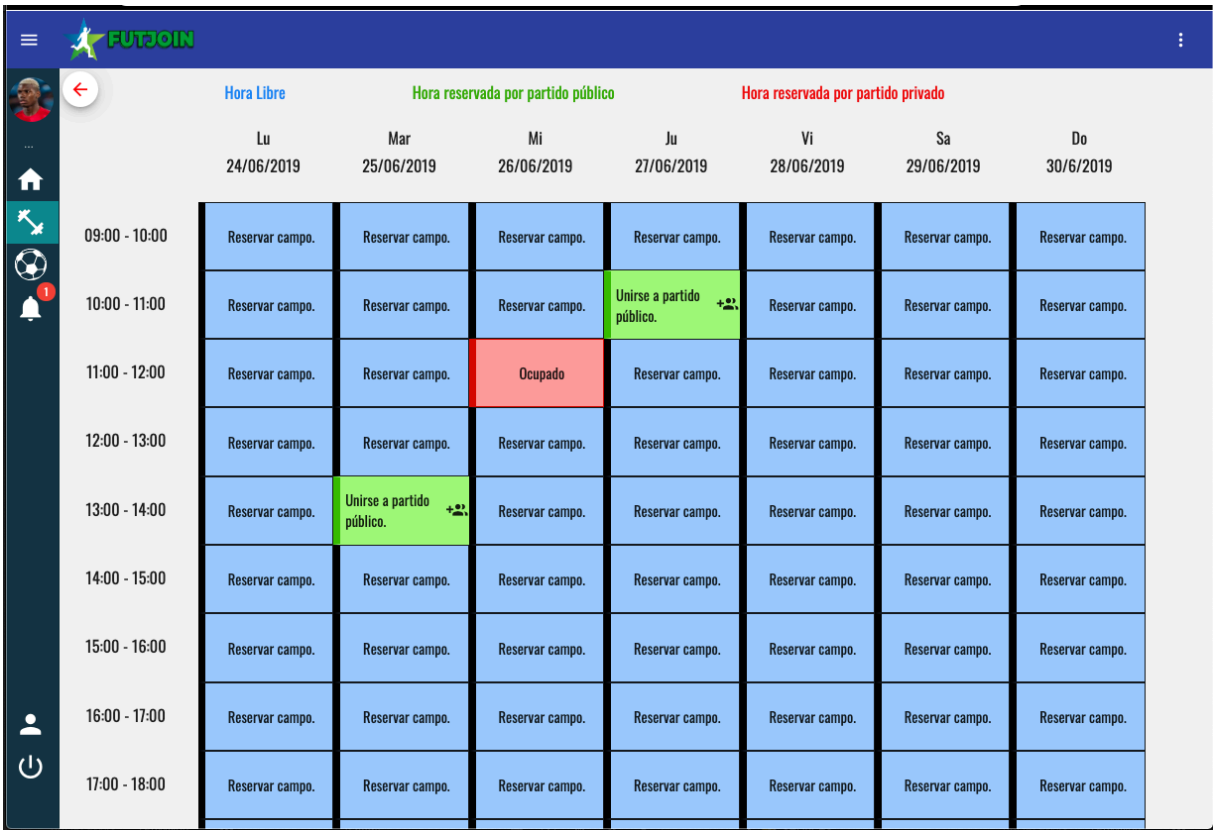
4. Campos

1. Pulsar el botón “Fútbol 7”, “Fútbol 11” o “Fútbol Sala” para visualizar los campos de ese tipo que pertenecen al complejo deportivo.

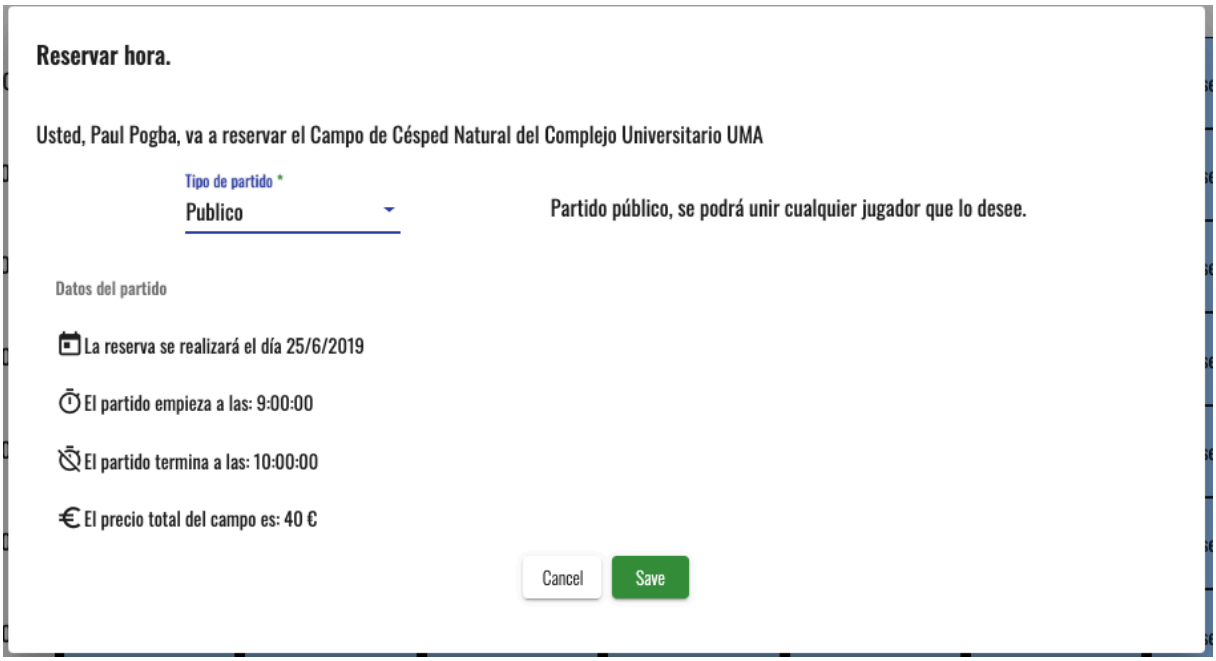
2. Pulsar sobre la fila de la tabla correspondiente al campo del que se quiere obtener información.

5. Partidos

5.1 Crear un partido

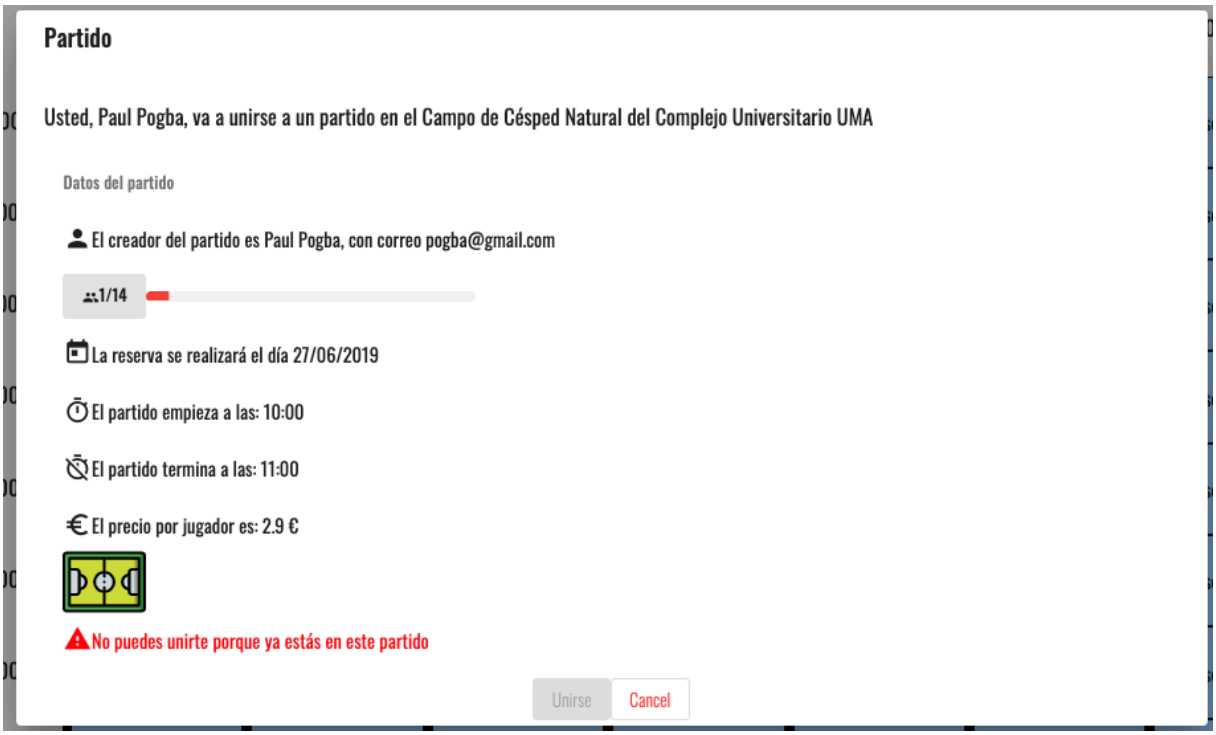


1. Pulsar sobre el botón de la pesa de la barra vertical izquierda.
2. Pulsar el botón “Fútbol 7”, “Fútbol 11” o “Fútbol Sala” para visualizar los campos de ese tipo que pertenecen al complejo deportivo.
3. Pulsar en “Ver reservas” en el campo elegido.
4. Elegir “Reservar campo” en el horario que lo permita.
5. Elegir el tipo de partido que se quiere crear.
6. Pulsar “Crear partido”.

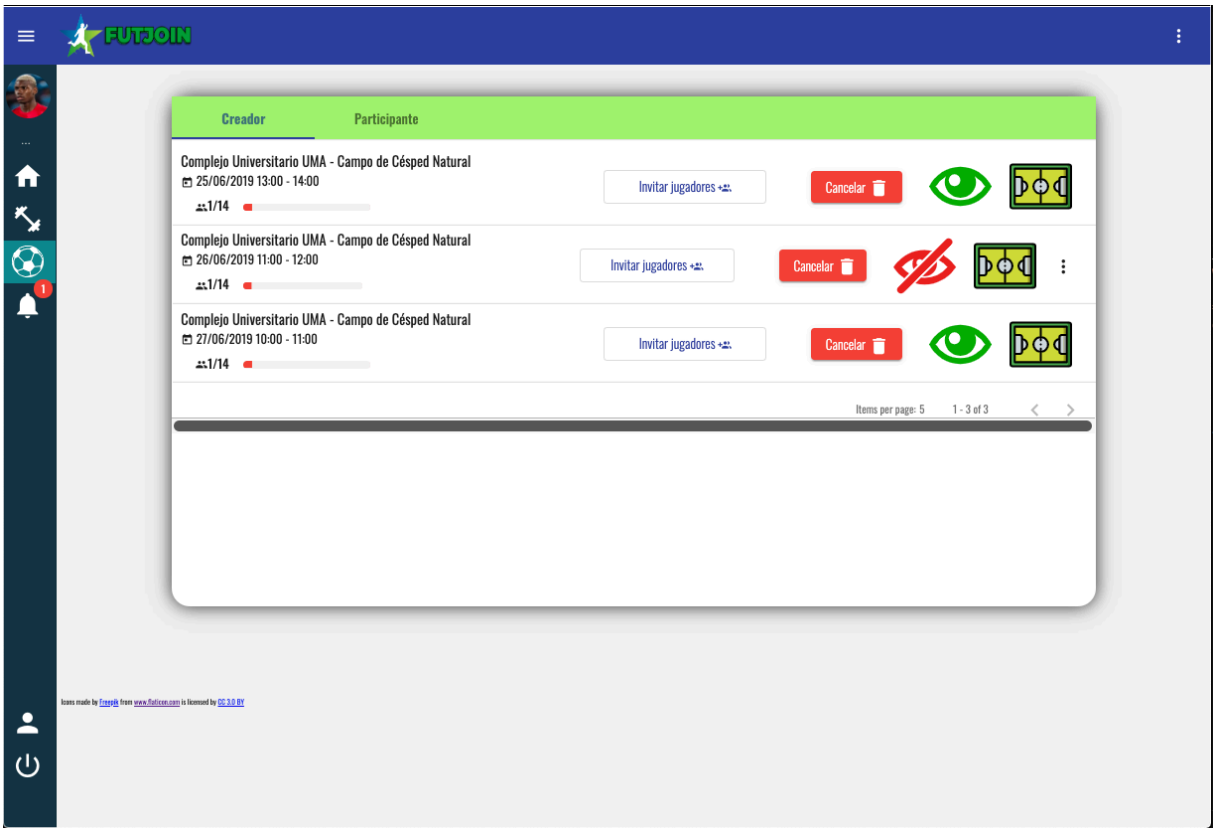


5.2 Unirse a partido público

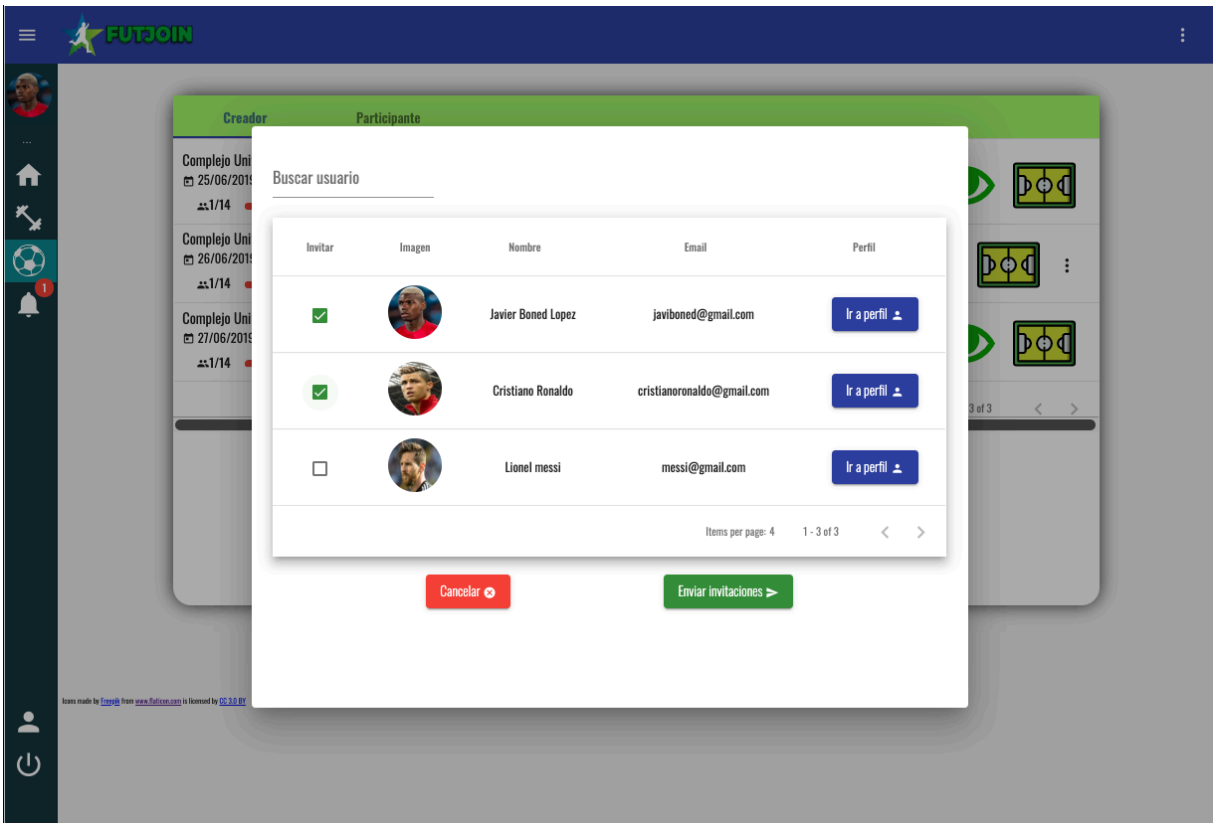
1. Pulsar sobre el botón de la pesa de la barra vertical izquierda.
2. Pulsar el botón “Fútbol 7”, “Fútbol 11” o “Fútbol Sala” para visualizar los campos de ese tipo que pertenecen al complejo deportivo.
3. Pulsar en “Ver reservas” en el campo elegido.
4. Elegir “Unirse a partido público” en el horario que lo permita.
5. Pulsar en “Unirse”.



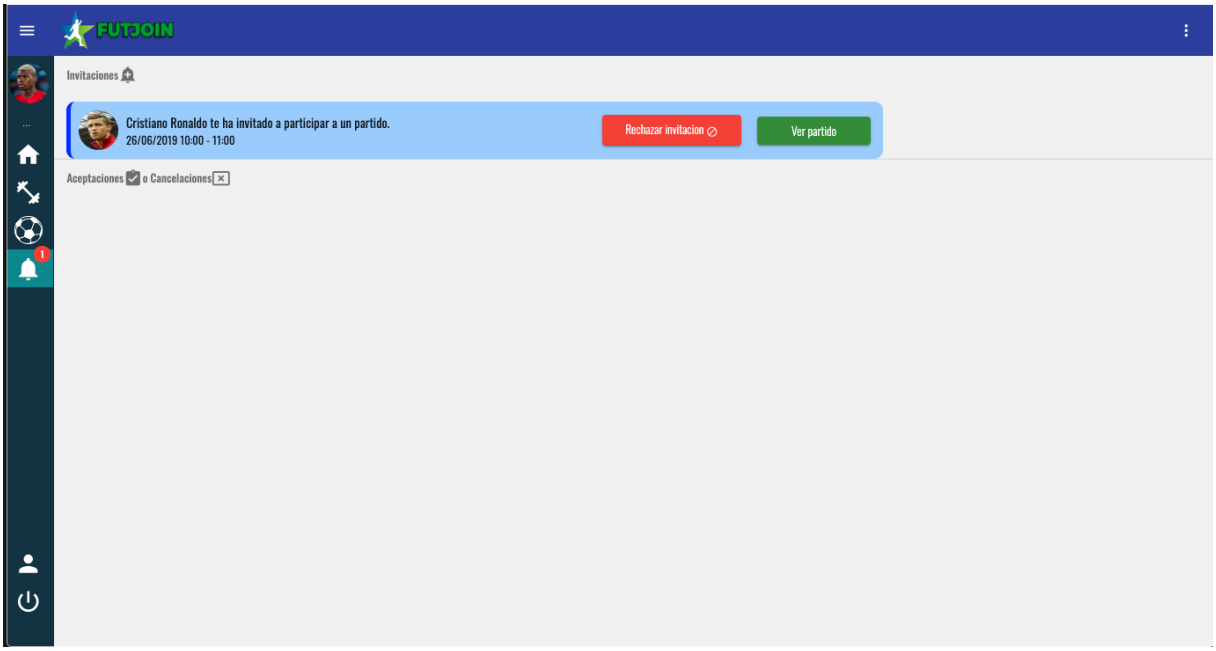
5.3 Invitar a jugadores a partido.



1. Pulsar sobre el botón del balón de la barra vertical izquierda.
2. En el apartado “creador” pulsar sobre el botón “invitar jugadores”.
3. Elegir los jugadores registrados que se quieren invitar.
4. Pulsar “Enviar Invitaciones”



5.4 Unirse a partido con invitación.



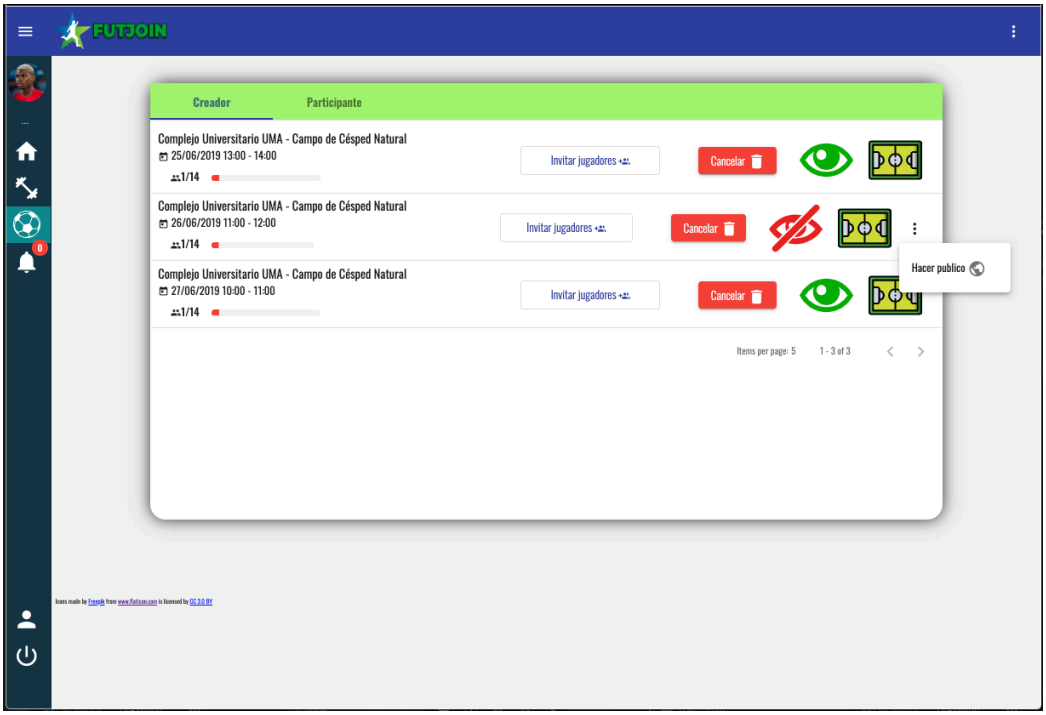
1. Pulsar sobre el botón de la campana de la barra vertical izquierda.
2. Pulsar en “Ver partido” en la invitación recibida.
3. Pulsar en “Unirse”

5.5 Convertir partido privado en público.

1.Pulsar sobre el botón del balón de la barra vertical izquierda.

2. En el apartado “creador” pulsar en el botón de los 3 botones verticales de un partido privado.

3. Pulsar en “Hacer público”.



5.6 Cancelar partido

1. Pulsar sobre el botón del balón de la barra vertical izquierda.

2. En el apartado “creador” pulsar sobre el botón “cancelar” sobre el partido que se desea cancelar.

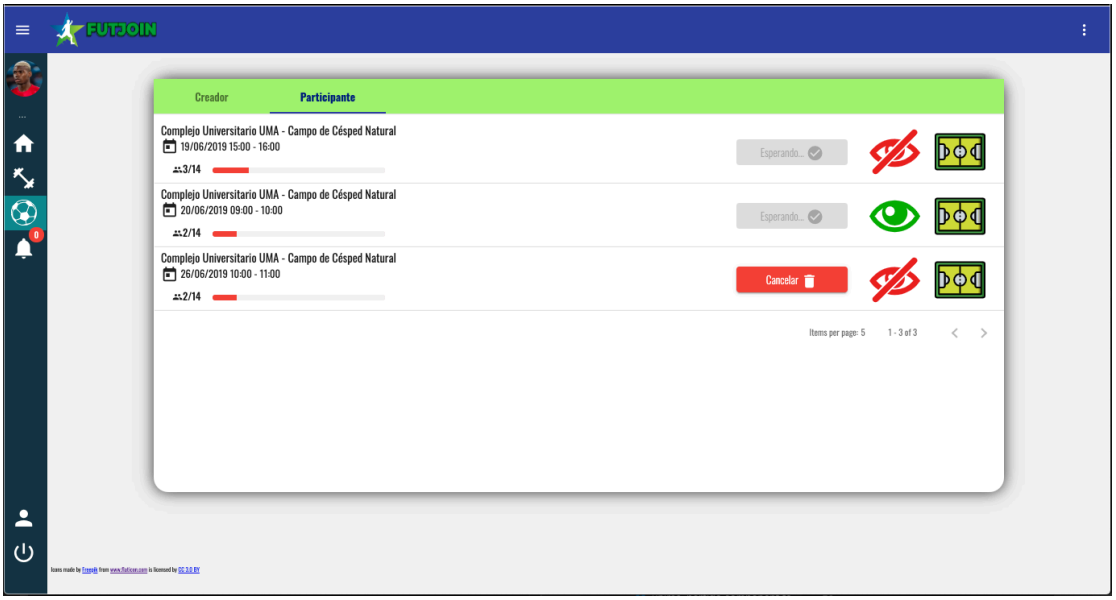
3. Pulsar “Si”.

5.7 Cancelar asistencia a un partido

1. Pulsar sobre el botón del balón de la barra vertical izquierda.

2. En el apartado “participante” pulsar sobre el botón “cancelar” sobre el partido que se desea cancelar.

3. Pulsar “Si”.



Apéndice B

Manual de Instalación

1. Requerimientos:

- PC con el sistema operativo Linux, Windows o Mac OSX.
- Tener instalado y ejecutnado *MongoDB*. https://docs.mongodb.com/manual/installation/
- Tener instalado *Node.js*. https://nodejs.org/es/download/
- Tener instalado *Angular*. https://cli.angular.io/
- Tener instalado el gestor de paquetes *npm*. https://www.npmjs.com/get-npm
- Navegador web (recomendado *Google Chrome*).

2. Instalación de servidor.

- Dentro de la carpeta "Backend" entregada, ejecutar el siguiente código: "npm install". Una vez que termine, bastará con ejecutar "npm start" para que el servidor se ejecute (normalmente) en el puerto 3000 de la máquina local.

3. Instalación de cliente.

- Dentro de la carpeta “Frontend” entregada, ejecutar el siguiente código: “npm install”. Una vez que termine, bastará con ejecutar “ng serve” para que el cliente se ejecute (normalmente) en el puerto 4200 de la máquina local.

- IMPORTANTE: Para el correcto funcionamiento del cliente, una vez ejecutado el comando “npm install”, se debe copiar y pegar la carpeta “ngx-carousel-3d” (entregada en la carpeta “Otros” del CD) en la carpeta “node_modules” creada en la carpeta “Frontend”.